

Keyifli Okumalar Dileriz

Binlerce kategoride milyonlarca PDF e-kitap, ders kitapları, dergiler ve romanlara platformumuz üzerinden ücretsiz ulaşabilirsiniz.

RESMİ WEB SİTEMİZ

<https://rantey.org>

Yedek ve Duyuru Kanallarımız

Sitemize erişim engeli gelmesi veya adres değişikliği durumunda aşağıdaki kanallardan güncel giriş adresine erişebilirsiniz:

- Telegram: <https://t.me/birkitapbir>
- WhatsApp Kanalımız : <https://whatsapp.com/channel/0029VbDHv8uE50Us4lvMoc0Y>

İçindekiler

Konu	Tarih	Kategori	Sayfa
Stored Procedure Yardımıyla Yeni Bir Kayıt Ekleme	08.11.2003	Ado.Net	4
Web Sayfalarında Stored Procedure Kullanımı	12.11.2003	Asp.Net	9
Stored Procedure Yardımıyla Tablodan Kayıt Silmek	12.11.2003	Ado.Net	16
Overload Metodların Gücü	13.11.2003	C#	24
Transaction Kavramı	17.11.2003	Ado.Net	31
Distributed Transactions	19.11.2003	Ado.Net	39
Bir Form ve Kontrollerinin Elle Programlanması	21.11.2003	C#	52
Params Anahtar Sözcüğünün Kullanımı	30.11.2003	C#	59
DataSet ve WriteXml Metodunun Kullanımı	30.11.2003	Ado.Net	65
Enumerators	01.12.2003	C#	68
Basit Bir Web Service Uygulaması	01.12.2003	Web Servis	72
DataTable Sınıfını Kullanarak Programatik Olarak Tablolar Oluşturmak-1	04.12.2003	Ado.Net	83
Struct (Yapı) Kavramı ve Class (Sınıf) ile Struct (Yapı) Arasındaki Farklar	04.12.2003	C#	88
DataTable Sınıfını Kullanarak Programatik Olarak Tablolar Oluşturmak-2	05.12.2003	Ado.Net	95
DataColumn.Expression Özelliği İle Hesaplanmış Alanların Oluşturulması	06.12.2003	Ado.Net	99
İlişkili Tabloları DataSet İle Kullanmak - 1	09.12.2003	Ado.Net	102
İlişkili Tabloları DataSet İle Kullanmak - 2	10.12.2003	Ado.Net	106
SQL DMO İşlemleri	12.12.2003	Ado.Net	111
DataGridView Sınıfı ve Faydaları	15.12.2003	Ado.Net	118
DataGridView Denetimi Üzerinde Sayfalama(Paging) İşlemi	16.12.2003	Ado.Net	124
DataGridView Denetimi Üzerinde Sıralama(Sorting) İşlemi	17.12.2003	C#	128
HashTable Koleksiyon Sınıfı	18.12.2003	C#	132
Stack ve Queue Koleksiyon Sınıfı	19.12.2003	C#	135
Reflection Sınıfı İle Tiplerin Sırrı Ortaya Çıkıyor	22.12.2003	C#	140
Bir Sınıf Yazalım	23.12.2003	C#	148
Virtual(Sanal) Metodlar	25.12.2003	C#	155
Kalıtım (Inheritance) Kavramına Kısa Bir Bakış	25.12.2003	C#	160
SqlDataReader Sınıfı 1	28.12.2003	Ado.Net	167
SqlDataReader Sınıfı 2	29.12.2003	Ado.Net	172
Boxing (Kutulamak) ve Unboxing (Kutuyu Kaldırmak)	30.12.2003	Ado.Net	178
XmlDataDocument Yardımıyla Xml Verilerini DataSet'e Aktarmak	30.12.2003	C#	
Çok Kanallı(Multithread) Uygulamalar	01.01.2004	C#	184
StreamReader Sınıfı Yardımıyla Dosya Okumak	01.01.2004	Ado.Net	189
Thread'leri Belli Süreler Boyunca Uyumak ve Yoketmek	02.01.2004	C#	192
Thread'lerde Öncelik(Priority) Durumları	05.01.2004	C#	198
İşe Yarar Bir MultiThreading(Çok Kanallı) Uygulama Örneği	06.01.2004	C#	206
ArrayList Koleksiyonu ve DataGridView	07.01.2004	C#	211
Interface (Arayüz) Kullanımına Giriş	08.01.2004	C#	219
Arayüz(Interface), Sınıf(Class) ve Çoklu Kalıtım	09.01.2004	C#	225
Arayüzler'de is ve as Anahtar Sözcüklerinin Kullanımı	12.01.2004	C#	229
Bir Arayüz, Bir Sınıf ve Bir Tablo	14.01.2004	C#	233
Checked, Unchecked Anahtar Kelimeleri ve Overflow Hatası	15.01.2004	C#	241
Temsilciler (Delegates) Kavramına Giriş	20.01.2004	C#	246
Net Data Providers(Veri Sağlayıcıları)	22.01.2004	Ado.Net	252
Sql Tablolarına Resim Ekleme	23.01.2004	Ado.Net	255
Sql Tablolarındaki Binary Resimlere Bakmak ve Dosya Olarak Kaydetmek	24.01.2004	Ado.Net	259
İndeksleyiciler (Indexers)	27.01.2004	C#	264
Tablo Değişikliklerini GetChanges ile İzlemek	29.01.2004	Ado.Net	270
Stored Procedureler ve ParameterDirection Numaralandırıcısı	31.01.2004	Ado.Net	276
Strongly Typed DataSet - 1 (Kuvvetle Türlendirilmiş Veri Kümeleri)	04.02.2004	Ado.Net	283

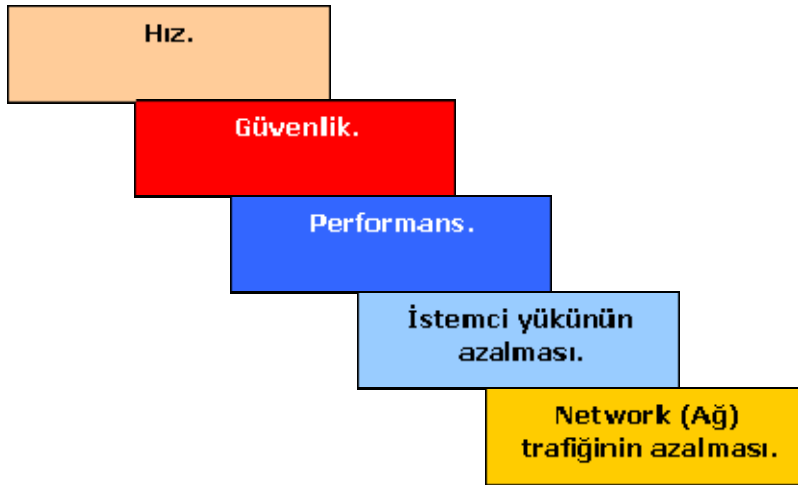
Strongly Typed DataSet - 2 (Kuvvetle Türlendirilmiş Veri Kümeleri)	05.02.2004	Ado.Net	292
Data Form Wizard Yardımıyla İlişkili Tablo Uygulamalarının Hazırlanması	09.02.2004	Ado.Net	300
GetOleDbSchemaTable Metodu İle Veritabanımızda Ne Var Ne Yok	12.02.2004	Ado.Net	316
Connection (Bağlantı) Kavramı ve OleDbConnection Sınıfı	13.02.2004	Ado.Net	322
Batch Queries (Toplu Sorgular) ve SqlDataReader	17.02.2004	Ado.Net	335
Command Kavramı ve OleDbCommand Sınıfı	23.02.2004	Ado.Net	339
DataAdapter Kavramı ve OleDbDataAdapter Sınıfına Giriş	27.02.2004	Ado.Net	354
OleDbDataAdapter Sınıfı - 2	02.03.2004	Ado.Net	369
OleDbDataAdapter Sınıfı ve Update Metodu.	14.03.2004	Ado.Net	384
OleDbDataAdapter Sınıfı Olayları	18.03.2004	Ado.Net	391
DataTable.Compute Metodu	29.03.2004	Ado.Net	401
DataRelation Sınıfı ve Çoğa-Çok (Many-to-many) İlişkiler	01.04.2004	Ado.Net	406
İlişkiler ve Hesaplanmış Alanların Bir Arada Kullanılması	08.04.2004	Ado.Net	415
Private Assembly ve Shared Assembly Kavramı	22.04.2004	Framework	418
Single File Assembly ve Multiple-File Assembly Kavramları	27.04.2004	Framework	426
Windows Servislerine Giriş	29.04.2004	Win Servis	437
Windows Servislerinin Kontrolü - 1	05.05.2004	Win Servis	449
Windows Servislerinin Kontrolü - 2 (Sistemdeki Servislerin Kontrol Edilmesi)	12.05.2004	Win Servis	460
.NET Remoting'i Kavramak	22.05.2004	Remoting	470
XML Rapor Web Servisleri	10.06.2004	Web Servis	476
Transaction' larda SavePoint Kullanımı	15.06.2004	Ado.Net	481
Transaction' larda İzolasyon Seviyeleri (Isolation Level) - 1	19.06.2004	Ado.Net	488
Transaction' larda İzolasyon Seviyeleri -2 (IsolationLevel Numaralandırıcısı)	28.06.2004	Ado.Net	497
NET Remoting' i Kavramak - 2	03.07.2004	Remoting	506
Transaction' larda DeadLock Kavramı	07.07.2004	Ado.Net	512
NET Remoting' i Kavramak - 3	22.07.2004	Remoting	517
CurrencyManager ile Navigasyon ve Temel Satır İşlemleri	27.07.2004	Ado.Net	523
Identity Değerlerinin Çalışma Zamanında Elde Edilmesi	29.07.2004	Ado.Net	530
Localization (Yerelleştirme) - 1	05.08.2004	C#	537
Localization (Yerelleştirme) 2 - Dil Desteği	07.08.2004	C#	544
Asp.Net 2.0 ile Cross-Page Posting	31.08.2004	Asp.Net	550
Asp.Net 2.0 ve Code Klasörü	01.09.2004	Asp.Net	557
Asp.Net 2.0 ile Veri Kümelerinde Sayfalama İşlemleri	01.09.2004	Asp.Net	564
Asp.Net 2.0 için Site Map Kullanımı	03.09.2004	Asp.Net	568
Asp.Net 2.0 ve Temalar (Themes)	03.09.2004	Asp.Net	575
Asp.Net 2.0 ve TreeView Kontrolü	06.09.2004	Asp.Net	581
Asp.Net 2.0 GridView Kontrolünde Update,Delete İşlemleri	07.09.2004	Asp.Net	589
Asp.Net 2.0 DetailsView Kontrolü ile Insert,Update,Delete	08.09.2004	Asp.Net	598
Asp.Net 2.0 ve Master Page Kavramı	10.09.2004	Asp.Net	603
Asp.Net 2.0 ve ObjectDataSource Kontrolü	14.09.2004	Asp.Net	612
Ado.Net 2.0 ve Bulk-Data Kopyalama Mekanizması	17.09.2004	Ado.Net	620
Ado.Net 2.0 ve Toplu Güncelleme İşlemleri (Batch-Updates)	18.09.2004	Ado.Net	632
Ado.Net 2.0 ile Mars' a Ayak Basıyoruz	20.09.2004	Ado.Net	637
Ado.Net 2.0 ve Sql Komutlarını Asenkron Olarak Yürütmek - 1	23.09.2004	Ado.Net	646
Ado.Net 2.0 ve Sql Komutlarını Asenkron Olarak Yürütmek - 2	25.09.2004	Ado.Net	654
Xml Web Servislerine Giriş - 1	29.09.2004	Web Servis	658
Xml Web Servislerine Giriş - 2	30.09.2004	Web Servis	667
Xml Web Servisleri - 3 (Mimarinin Temelleri - SOAP)	01.10.2004	Web Servis	677
Xml Web Servisleri - 4 (Mimarinin Temelleri - WSDL)	02.10.2004	Web Servis	687
Xml Web Servisleri - 5 (Mimarinin Temelleri - DISCO)	07.10.2004	Web Servis	700
Ado.Net 2.0 ve Sql Komutlarını Asenkron Olarak Yürütmek - 3	22.10.2004	Ado.Net	704
Ado.Net 2.0 ve SqlDependency Sınıfı Yardımıyla Query Notification	26.10.2004	Ado.Net	708
Oyun Programlamaya Giriş (Çarpışma Teknikleri - 1)	06.11.2004	C#	713
Oyun Programlamaya Giriş (Çarpışma Teknikleri - 2)	12.11.2004	C#	720
Oyun Programlamaya Giriş (Çarpışma Teknikleri - 3)	19.11.2004	C#	726

Oyun Programlamaya Giriş (Matrisler Yardımıyla Çarpışma Kontrolü)	04.12.2004	C#	733
Ado.Net ile Sql Server Full-Text Searching (Tüm Metinde Arama) Kullanımı	18.12.2004	Ado.Net	744
Derinlemesine Session Kullanımı – 1	31.12.2004	Asp.Net	756
Derinlemesine Session Kullanımı – 2	08.01.2005	Asp.Net	764
Caching Mekanizmasını Anlamak - 1	21.01.2005	Asp.Net	777

Stored Procedure Yardımıyla Yeni Bir Kayıt Ekleme

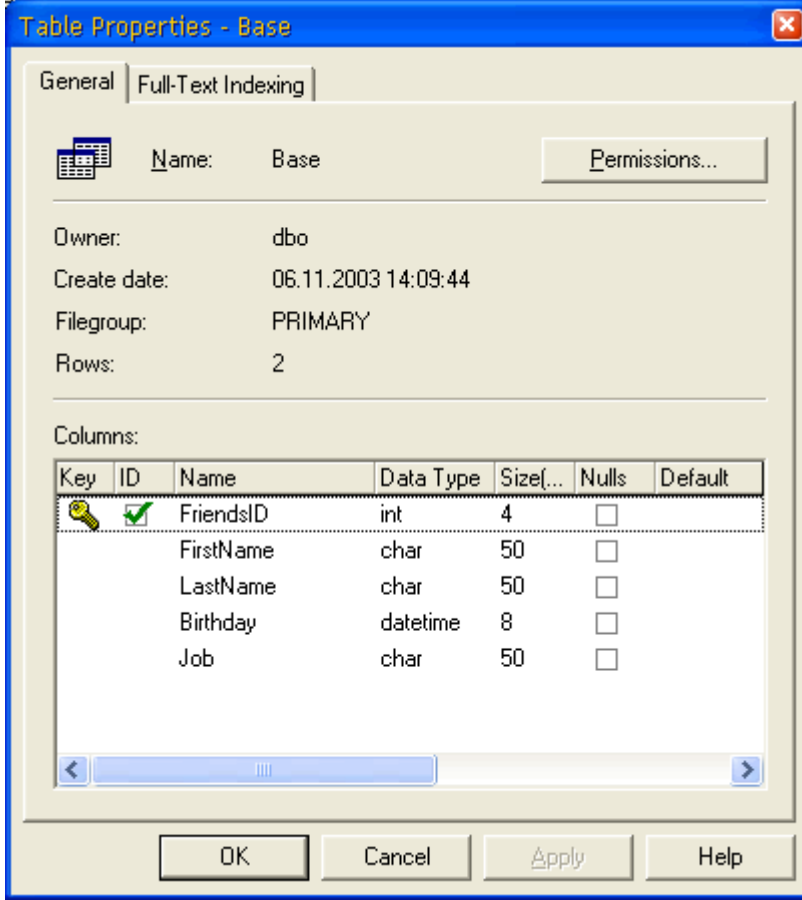
Bu yazımızda Sql Server üzerinde, kendi yazdığımız bir Saklı Yordam (Saklı Yordam) ile , veritabanındaki ilgili tabloya nasıl kayıt ekleyeceğimizi incelemeye çalışacağız.

Öncelikle, Saklı Yordamlar hakkında kısa bir bilgi vererek hızlı bir giriş yapalım. Saklı yordamlar derlenmiş sql cümlecikleridir. Bunlar birer veritabanı nesnesi oldukları için, doğrudan veritabanı yöneticisi olan programda (örneğin Sql Server) yer alırlar. Bu nedenle veritabanınızı bir yere taşıdığınızda otomatik olarak, saklı yordamlarınızda taşımış olursunuz. Bu Saklı Yordam'lerin tercih edilme nedenlerinden sadece birisidir. Diğer yandan, derlenmiş olmaları aslında bu sql cümleciklerinin doğrudan makine diline dönüştürüldüğü anlamına gelmez. Aslında , çalıştırmak istediğimiz sql cümleciklerini bir Saklı Yordam içine yerleştirerek, bunun bir veritabanı nesnesi haline gelmesini ve çalıştırıldığında doğrudan, veritabanı yöneticisini üzerinde barındıran sunucu makinede işlemlerini sağlarız. Bu doğal olarak, istemci makinelerdeki iş yükünü azaltır ve performansı artırır. Nitekim bir program içinde çalıştırılan sql cümleleri, Saklı Yordam' lardan çok daha yavaş sonuç döndürür. Dolayısıyla Saklı Yordamlar özellikle çok katlı mimariyi uygulamak istediğimiz projelerde faydalıdır. Saklı Yordamların faydalarını genel hatları ile özetlemek gerekirse ;



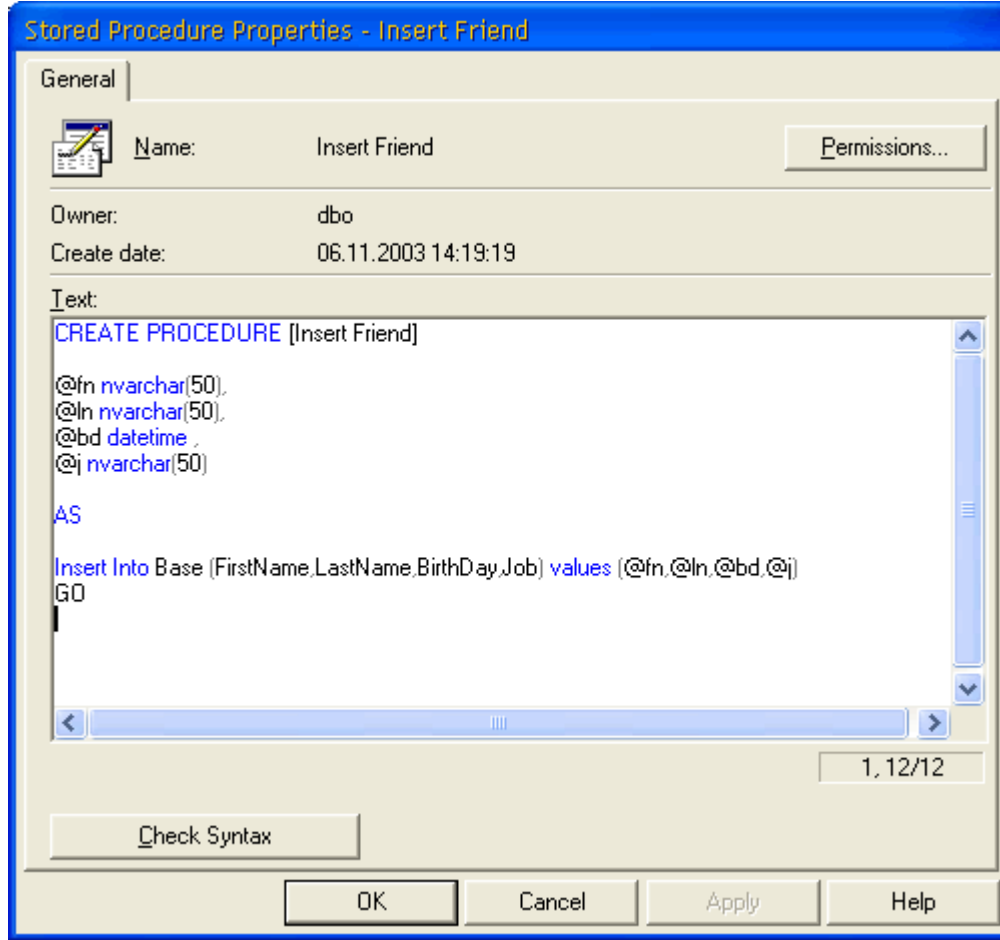
Şekil 1. Saklı Yordam Kullanmanın Avantajları.

İşte bizim bugünkü uygulamamızda yapacağımız işlemde budur. Bu uygulamamızda basit bir Saklı Yordam yaratacak, SqlCommand nesnesinin CommandType özelliğini, SqlParameter koleksiyonunu vb. kullanarak geniş bir bilgi sahibi olacağız. Öncelikle üzerinde çalışacağımız tablodan bahsetmek istiyorum. Basit ve konuyu hızlı öğrenebilmemiz açısından çok detaylı bir kodlama tekniği uygulamıyacağım. Amacımız Saklı Yordamımıza parametreler göndererek doğrudan veritabanına kaydetmek olacak. Dilerseniz tablomuzu inceleyelim ve oluşturalım.



Şekil 2. Tablonun Yapısı.

Şekil 2' de tablomuzda yer alan alanlar görülmekte. Bu tabloda arkadaşlarımızın doğum günlerini, işlerini , isim ve soyisim bilgilerini tutmayı planlıyoruz. Tablomuzda FriendsID isminde Primary Key olan ve otomatik olarak artan bir alanda yer alıyor. Şimdi ise insert sql deyimini kullandığımız Saklı Yordamımıza bir göze atalım.



Şekil 3. Insert Friend Saklı Yordamının Kodları.

Şekil 3 kullanacağımız Saklı Yordamın T-SQL(Transact SQL) deyimlerini gösteriyor. Burada görüldüğü gibi Sql ifademizin 4 parametresi var. Bu parametrelerimiz;

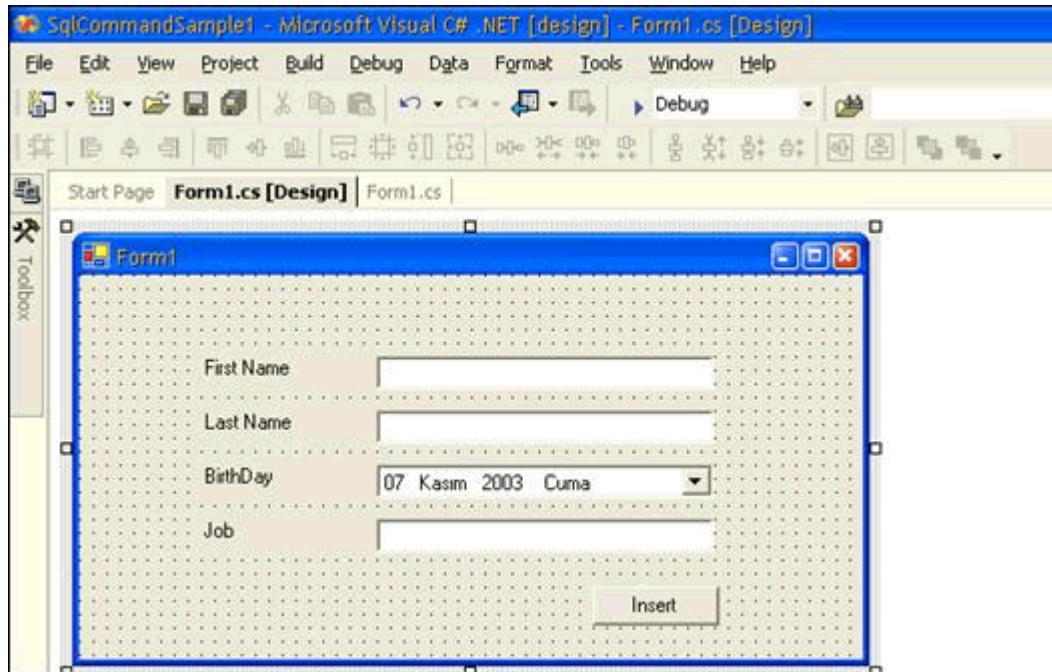
Parametre Adı	Veri Tipi	Veri Uzunluğu	Açıklama
@fn	Nvarchar	50	First Name alanı için kullanılacak.
@ln	Nvarchar	50	Last Name alanı için kullanılacak.
@bd	Datetime	-	BirthDay alanı için kullanılacak.
@j	Nvarchar	50	Job alanı için kullanılacak.

Tablo 1. Saklı Yordamımızda Kullanılan Giriş Parametreleri.

Insert Into Base (FirstName,LastName,BirthDay,Job) values (@fn,@ln,@bd,@j)

cümlecığı ile standart bir kayıt ekleme işlemi yapıyoruz. Tek önemli nokta values(değerler) olarak, parametre değerlerini gönderiyor olmamız. Böylece, Saklı Yordamımız, .net uygulamamızdan alacağı parametre değerlerini bu sql cümlecığıne alarak, tablomuz üzerinde yeni bir satır oluşturulmasını sağlıyor. Peki bu parametre değerlerini .net uygulamamızdan nasıl vereceğiz? Bunun için uygulamamızda bu Saklı Yordamı kullanan bir SqlCommand nesnesi oluşturacağız. Daha sonra, Saklı Yordamımızda yer alan parametreleri, bu SqlCommand nesnesi için oluşturacak ve Parameters koleksiyonuna ekleyeceğiz. Bu işlemin tamamlanmasının ardından tek yapacağımız Saklı Yordama geçecek parametre değerlerinin, SqlCommand nesnesindeki uygun SqlParameter nesnelere aktarılması ve Saklı Yordamın çalıştırılması olacak.

Öncelikle C# için yeni bir Windows Application oluşturalım ve formumuzu aşağıdaki şekilde düzenleyelim. Burada 3 adet textBox nesnemiz ve tarih bilgisini girmek içinde bir adet DateTimePicker nesnemiz yer alıyor. Elbette insert işlemi içinde bir Button kontrolü koymayı ihmal etmedik. Kısaca formun işleyişinden bahsetmek istiyorum. Kullanıcı olarak biz gerekli bilgileri girdikten sonra insert başlıklı Button kontrolüne bastığımızda, girdiğimiz bilgiler Saklı Yordam' daki parametre değerleri olacak. Ardından Saklı Yordamımız çalıştırılacak ve girdiğimiz bu parametre değerleri ile, sql sunucumuzda yer alan veritabanımızdaki Base isimli tablomuzda yeni bir satır oluşturulacak.



Şekil 4. Formun Tasarım Zamanındaki Görüntüsü.

Şimdide kodumuzu inceleyelim. Her zaman olduğu gibi **SQLClient** sınıfına ait nesneleri kullanacağımız için bu sınıfı **using** ile projemizin en başına ekliyoruz.

```
using System;
using System.Drawing;
using System.Collections;
```



```
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;
```

Sırada veritabanına olan bağlantımızı referans edicek olan **SqlConnection** nesnemiz var.

```
SqlConnection conFriends = new SqlConnection("initial catalog=Friends;data source=localhost;integrated security=sspi;packet size=4096");
```

Kısaca anlatmak gerekirse, SQL Sunucumuz' daki Friends isimli Database' e bağlantı sağlayacak bir **SqlConnection** nesnesi tanımladık. Burada **SqlConnection** sınıfının prototipi aşağıda verilen **Constructor**(yapıcı metodunu) metodunu kullandık. Bildiğiniz gibi SqlConnection nesnesi, Sql Sunucusu ile ado.net nesneleri arasında iletişimin sağlanabilmesi için bir bağlantı hattı tesis etmektedir.

```
public SqlConnection(string connectionString);
```

Şimdi btnInsert isimli butonumuzun click olay procedure' ündeki kodumuzu yazalım.

```
private void btnInsert_Click(object sender, System.EventArgs e)
{
    conFriends.Open();/* Baglanti açılıyor. SqlCommand nesnesi ile ilgili ayarlamalara geçiliyor. Komut SQL Server' da Friends database'inde yazılı olan "Insert Friend" isimli Saklı Yordam'ı çalıştıracak. Bu Procedure' ün ismini, CommandText parametresine geçirdikten sonar ikinci parameter olarak SqlConnection nesnemizi belirtiyoruz.*/
    SqlCommand cmdInsert = new SqlCommand("Insert Friend",conFriends);
    /* SqlCommand nesnesinin CommandType degerinide CommandType.StoredProcedure yapıyoruz. Bu sayede CommandText'e girilen degerin bir Saklı Yordam'e işaret ettiğini belirtmiş oluyoruz.*/
    cmdInsert.CommandType=CommandType.StoredProcedure;
    /* Şimdi bu Saklı Yordam için gerekli parametreleri oluşturalım. Bunun için SqlCommand nesnesinin parameters koleksiyonunun Add metodunu kullanıyoruz. Parametreleri eklerken, parametre isimlerinin SQL Server'da yer alan Saklı Yordamlardaki parametre isimleri ile aynı olmasına ve başlarına @ isareti gelmesine dikkat ediyoruz. Bu Add metodunun ilk parametresinde belirtiliyor. Add metodu ikinci parametre olarak bu parametrenin veri tipini alıyor. Üçüncü parametresi ise bu parametrik degiskenin boyutu oluyor.*/
    SqlParameter paramFirstName=cmdInsert.Parameters.Add("@fn",SqlDbType.NVarChar,50);
    /* Burada SqlCommand nesnesine @fn isimli nvarchar tipinde ve uzunluğu 50 karakterden oluşan bir parametre ekleniyor. Aynı şekilde diğer parametrelerimizi de belirtiyoruz.*/
    SqlParameter paramLastName=cmdInsert.Parameters.Add("@ln",SqlDbType.NVarChar,50);
    SqlParameter paramBirthDay=cmdInsert.Parameters.Add("@bd",SqlDbType.DateTime);
    SqlParameter paramJob=cmdInsert.Parameters.Add("@j",SqlDbType.NVarChar,50);
    // Şimdide parametrelerimize degerlerini verelim.
```

```

paramFirstName.Value=txtFirstName.Text;
paramLastName.Value=txtLastName.Text;
paramBirthDay.Value=dtBirthDay.Text;
paramJob.Value=txtJob.Text;
// Böylece ilgili parametrelere degerleri geçirilmis oldu. simdi komutu calistiralim.
cmdInsert.ExecuteNonQuery();
/* Böylece Saklı Yordamimiz, parametrelerine atanan yeni degerler ile calisitirilir. Bunun sonucu olarak SQL Server' daki Saklı
Yordama burada belirttiğimiz parametre deęerleri gider ve insert cümlecięi çalıştırılarak yeni bir kayıt eklenmis olur.*/
conFriends.Close(); // Son olarak SqlConnection' ımızı kapatıyoruz.
}

```

Şimdi bir deneme yapalım.

Şekil 5. Programın Çalışması.

FriendsID	FirstName	LastName	Birthday	Job
1000	Burak Selim	SENYURT	04.12.1976	Matematik Mühendi

Şekil 6. Saklı Yordam'ün işleminin Sonucu.

Görüldüğü gibi Saklı Yordamlar yardımıyla tablolarımıza veri eklemek son derece kolay, hızlı ve etkili. Bununla birlikte Saklı Yordamlar sağladıkları güvenlik kazanımları nedeni ile tercih edilirler. Saklı Yordamları geliştirmek son derece kolaydır. İstediklerini sql işlemini gerçekleştirebilirsiniz. Satır silmek, satır aramak gibi. Saklı Yordamlar ile ilgili bir sonraki makalemizde, tablolardan nasıl satır silbileceğimizi incelemeye çalışacağız. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Web Sayfalarında Stored Procedure Kullanımı

Bugünkü makalemde sizlere bir Web Sayfası üzerinde, bir tablonun belli bir satırına ait detaylı bilgilerin, bir Stored Procedure yardımıyla nasıl gösterileceğini anlatmaya çalışacağım. Uygulamamızda örnek olması açısından, Kitap bilgileri barındıran bir Sql tablosu kullanacağım. Tablomuzun yapısını aşağıdaki Şekil 1’ de görebilirsiniz. Temel olarak, kitap isimlerini, kitapların kategorilerini, yazar isimlerini, basım evi bilgilerini vb... barındıran bir tablomuz var. Bu tablonun örnek verilerini de Şekil 2’ de görebilirsiniz.

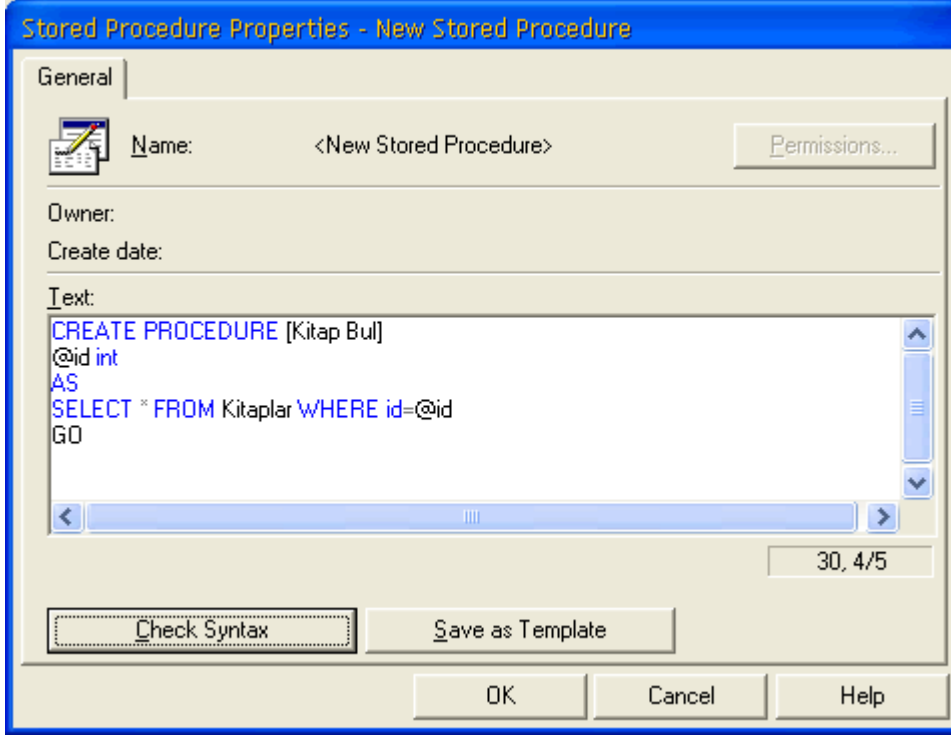
	Column Name	Data Type	Length	Allow Nulls
1	ID	int	4	
	Kategori	nvarchar	50	
	Adi	nvarchar	150	
	Yazar	nvarchar	100	
	BasimEvi	nvarchar	50	
	BasimTarihi	nvarchar	4	
	Resim	nvarchar	255	✓
	Link	nvarchar	150	✓
	Fiyat	money	8	✓

Şekil 1. Kitaplar tablosunun alan yapısı.

ID	Kategori	Adi	Yazar	BasimEvi	BasimTarihi	Resim	Link	Fiyat
1	Bilgisayar Programlama	Delphi 5'e Bakış	Ruhvar Barengi	SECKIN	2000	images/fk	<NULL>	<NULL>
3	Bilgisayar Programlama	Delphi 5 Uygulama Geliştirme Kılavuzu	Marco Cantu	ALFA	2000	images/fk	<NULL>	<NULL>
4	Bilgisayar Programlama	Delphi 5 Kullanım Kılavuzu	Dr.Cahit Akın	ALFA	2000	images/fk	<NULL>	<NULL>
5	Bilgisayar Programlama	Microsoft Visual Basic 6.0 Geliştirmek Üst	Teresa Canady, Pete Harris, Su	ARKADAS	2000	images/fk	http://ww	<NULL>
6	Bilgisayar Programlama	Visual Basic 6 Temel Başlangıç Kılavuzu	Greg Perry, Sanjya Hettihewa	SISTEM	2000	images/fk	http://ww	<NULL>
7	Bilgisayar Programlama	Microsoft Visual Basic 6 Temel Kullanım Ki	Faruk Çubukçu	ALFA	2001	images/fk	<NULL>	<NULL>
8	WEB Programlama	ASP ile E-Ticaret Programcılığı	Stephen Walker, Jonathan Levi	SISTEM	2001	images/fk	http://ww	<NULL>
9	WEB Programlama	ASP 3.0	Nicholas Chase	SISTEM	2001	images/fk	http://ww	<NULL>
10	WEB Programlama	ASP ile web programcılığı ve elektronik tic	Zafer Demirkol	PUSULA	2001	images/fk	http://ww	<NULL>
11	WEB Programlama	Herkes İçin ASP İle Veritabanı Yönetimi 3	Faruk Çubukçu	ALFA	2001	images/fk	<NULL>	<NULL>
12	Sistem Mühendisliği	Analiz, tasarım ve Uygulamalı Sistem Yön	Prof.Dr.Haluk Erkut	IRFAN	2000	images/fk	<NULL>	<NULL>
13	WEB Programlama	Adm Adım Web Veritabanı Geliştirme	Jim Buysens	ARKADAS	2000	images/fk	http://ww	<NULL>
14	WEB Programlama	Macromedia Flash 5 actionscript yaratıcı	Derek Franklin, brooks patton	SISTEM	2000	images/fk	http://ww	<NULL>
15	WEB Programlama	Flash 5 actionscript	Memduh Saraç	SECKIN	2000	images/fk	<NULL>	<NULL>
16	WEB Programlama	Macromedia Flash 4 Macintosh ve Windo	Kathrine Ulrich	SISTEM	1999	images/fk	http://ww	<NULL>
18	Bilgisayar Programlama	SQL Server 7 Sistem Yönetimi Ve Uygular	Faruk Çubukçu	ALFA	1999	images/fk	http://ww	2000C
19	Bilgisayar Genel Konular	Bilgisayar terimleri ansiklopedik sözlüğü	Naci Altan	SISTEM	2000	images/fk	http://ww	<NULL>
20	Bilgisayar Genel Konular	Internet Information Server	Murat Yıldırımoglu	PUSULA	2000	<NULL>	http://ww	<NULL>
21	WEB Programlama	Webmaster için Javascript	Numan Peköz	PUSULA	2000	images/fk	http://ww	<NULL>
22	Matematik Mühendisliği	Nümerik Analiz	Yrd.Doç.Dr. İbrahim Uzun	BETA	2000	images/fk	<NULL>	<NULL>
23	Matematik Mühendisliği	Çözümlü Soyut Cebir Programları	Prof.Dr. Fethi Çallıalp	BELLI DEGİL	2000	images/fk	<NULL>	<NULL>
24	Matematik Mühendisliği	Matematik Analiz 3	Doç. Dr. Cevdet Cerit	ITU	2000	images/fk	<NULL>	<NULL>
25	Matematik Mühendisliği	Matematik Analiz 4	Doç. Dr. Cevdet Cerit	ITU	2000	images/fk	<NULL>	<NULL>
26	Matematik Mühendisliği	Matematik Analiz Cilt 1	Mustafa Balcı	BALCI	2000	images/fk	<NULL>	<NULL>
27	Matematik Mühendisliği	Matematik Analiz Cilt 2	Mustafa Balcı	BALCI	2000	images/fk	<NULL>	<NULL>

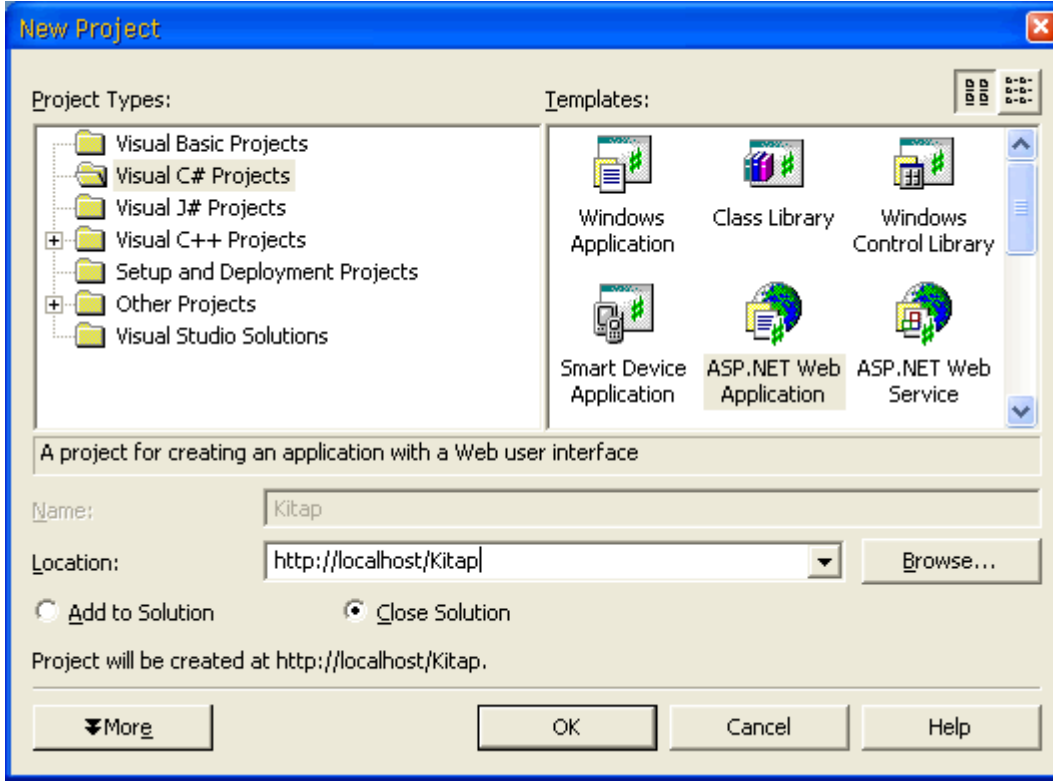
Şekil 2. Kitaplar tablosunun örnek verileri.

Şimdi projemizin en önemli unsuru olan Stored Procedure nesnemizi Sql Server üzerinde oluşturalım. Bu Stored Procedure ile kullanıcının, web sayfasında listbox nesnesi içinden seçtiği kitaba ait tüm verileri döndürecek olan bir Sql cümlecisi yazacağız. Burada aranan satırı belirleyecek olan değerimiz ID isimli aynı zamanda Primary Key olan alanın değeri olacaktır. Kullanıcı listBox nesnesinde yer alan bir kitabı seçtiğinde (yani listBox nesnesine ait IstKitaplar_SelectedIndexChanged olay procedure'ü çalıştırıldığında) seçili olan öğeye ait id numarası Stored Procedure'ümüze parametre olarak gönderilecek. Elde edilen sonuç kümesine ait alanlar dataGrid nesnemizde gösterilerek kitabımıza ait detaylı bilgilerin görüntülenmesi sağlanmış olacak. Dilerseniz "Kitap Bul" isimli Stored Procedure'ümüzü oluşturarak devam edelim. Şekil 3 yazdığımız Stored Procedure nesnesini göstermekte.



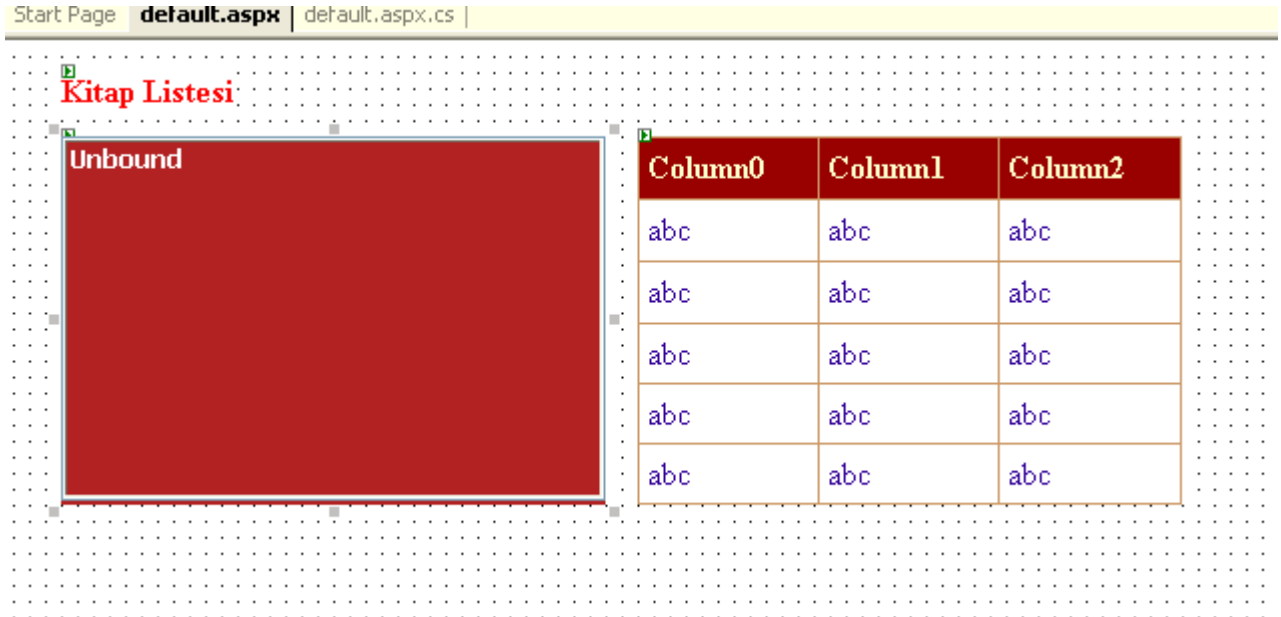
Şekil 3. Stored Procedure nesnemiz ve Sql ifadesi.

Sıra geldi uygulamamızda yer alan WebForm'umuzu oluşturmaya. Uygulamamızı C# dili ile oluşturmaya tercih ettiğimden New Project kısmında Visual C# Proejct bölümünü seçtim. Dikkat edecek olursanız, uygulmamız bir Web Application dır. Oluşturulduğu yer <http://localhost/kitap> adlı adrestir.



Şekil 4. Web Application

Evet gelelim WebFormun tasarımına. Ben aşağıdaki gibi bir tasarım oluşturdum. Sizlerde buna yakın bir tasarım oluşturabilirsiniz veya aynısını kullanmayı tercih edebilirsiniz.



Şekil 5. Web Form tasarımı.

Sıra geldi kodlarımızı yazmaya. Önce sayfa yüklenirken neler olacağını belirleyeceğimiz kodlarımızı yazmaya başlayalım. Özet olarak Page_Load olay procedure'ünde Sql Server ` a bağlanıp, Kitaplar tablosundan yalnızca ID ve Adi alanına ait değerleri alıyoruz ve bunları bir SqlDataReader nesnesi vasıtasıyla, IstKitaplar isimli listBox nesnemize yüklüyoruz. Gelin kodumuzu yazalım, hem de inceleyelim.

```
/* Önce gerekli SqlConnection nesnemizi oluşturuyor ve gerekli ayarlarımızı yapıyoruz.*/
```

```
SqlConnection conFriends=new SqlConnection("initial catalog=Friends;Data Source=localhost;integrated security=sspi");
```

```
private void Page_Load(object sender, System.EventArgs e)
```

```
{
```

```
    if (Page.IsPostBack==false)
```

```
    {
```

```
        /* SqlCommand nesnemizi yaratıyoruz. Bu nesne Select sorgusu ile Kitaplar tablosundan ID ve Adi alanlarının değerlerini alacak. Alınan bu veri kümesi Adi alanına göre A'dan Z'ye sıralanmış olacak. Bunu sağlayan sql cümleciğindeki, "Order By Adi" ifadesidir. Tersten sıralamak istersek "Order By Adi Asc" yazmamız gerekir. */
```

```
        SqlCommand cmdKitaplar=new SqlCommand("Select ID,Adi From Kitaplar Order By Adi",conFriends);
```

```
        cmdKitaplar.CommandType=CommandType.Text; // SqlCommand'in command String'inin bir Sql cümleciğine işaret ettiğini belirtiyoruz.
```

```
        SqlDataReader dr; // Bir SqlDataReader nesnesi oluşturuyoruz.
```

```
        /* SqlDataReader nesnesi ileri yönlü ve sadece okunabilir bir veri akışı sağlar. (Forward and Readonly) Bu da nesneden veri aktarımlarının (örneğin bir listBox'a veya datagrid'e) hızlı çalışmasına bir nedendir. Uygulamalarımızda, listeleme gibi sadece verilere bakmak amacıyla çalıştıracağımız sorgulamalar için, SqlDataReader nesnesini kullanmak, performans açısından olumlu etkiler yapar. Ancak SqlDataReader nesnesi çalıştığı süre boyunca sunucuya olan bağlantısında sürekli olarak açık olmasını gerektirir.
```

```
        Yukarıdaki kod satırında dikkat çekici diğer bir unsur ise, bir new yapılandırıcısı kullanılmayıdır. SqlDataReader sınıfının bir yapıcı metodu ( Constructor ) bulunmamaktadır. O nedenle bir değişken tanımlanmış gibi bildirilir. Bu nesneyi asıl yükleyen, SqlCommand nesnesinin ExecuteReader metodudur. */
```

```
        conFriends.Open(); // Bağlantımızı açıyoruz.
```

```
        dr=cmdKitaplar.ExecuteReader(CommandBehavior.CloseConnection); /* Burada ExecuteReader metodu , SqlCommand nesnesine şöyle bir seslenişte bulunuyor. " SqlCommand'cığım, sana verilen Sql Cümleciğini (Select sorgusu) çalıştır ve sonuçlarını bir zahmet eşitliğin sol tarafında yer alan SqlDataReader nesnesinin bellekte referans ettiği alana yükle. Sonrada sana belirttiğim, sağındaki CommandBehavior.CloseConnection parametresi nedeni ile, SqlDataReader nesnesi Close metodu ile kapatıldığında, yine bir zahmet SqlConnection nesnesinde otomatik olarak kapanmasını sağlayıver". */
```

```
        IstKitaplar.DataSource=dr; // Elde edilen veri kümesi SqlDataReader nesnesi yardımıyla ListBox nesnesine veri kaynağı olarak gösteriliyor.
```

```
        IstKitaplar.DataTextField="Adi"; // ListBox nesnesinde Text olarak Adi alanının değerleri görünülecek.
```

```
        IstKitaplar.DataValueField="ID"; // Görünen Adi değerlerinin sahip olduğu ID değerleri de ValueField olarak belirleniyor. Böylece "Kitap Bul" isimli Stored Procedure'ümüze ID parametresinin değeri olarak bu alanın değeri gitmiş olacak. Kısacası görünen yazı kitabın adı olurken, bu yazının değeri ID alanının değeri olmuş oluyor.
```

```
        IstKitaplar.DataBind(); // Web Sayfalarında , verileri nesnelere bağlarken DataBind metodu kullanılır.
```

```
        dr.Close(); // SqlDataReader nesnemiz kapatılıyor. Tabiki SqlConnection nesnemizde otomatik olarak kapatılıyor.
```

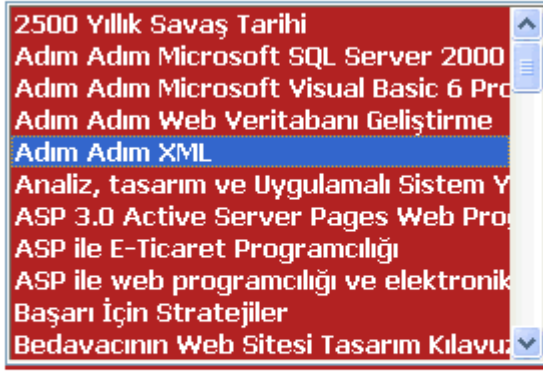
```

}
}

```

Şimdi oluşturduğumuz projeyi çalıştırsak aşağıdaki gibi bir sonuç elde ederiz. Görüldüğü gibi Kitaplar tablosundaki tüm kitaplara ait Adi alanlarının değerleri listBox nesnemize yüklenmiştir.

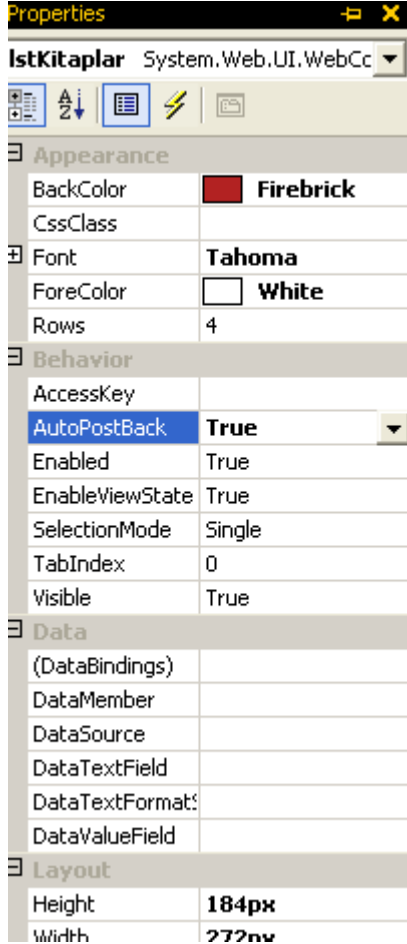
Kitap Listesi



Şekil 6. Page_Load sonrası.

Şimdi ise listBox'ta bir öğeyi seçtiğimizde neler olacağına bakalım. Temel olarak, seçilen öğeye ait ID değeri "Kitap Bul" isimli Stored Procedure'e gidicek ve dönen sonuçları dataGrid nesnesinde göstereceğiz.

ListBox nesnesine tıklandığı zaman, çalışacak olan lstKitaplar_SelectedIndexChanged olay procedure'ünde gerekli kodları yazmadan önce ListBox nesnesinin **AutoPostBack** özelliğine **True** değerini atamamız gerekiyor. Böylece kullanıcı sayfa üzerinde listbox içindeki bir nesneye tıkladığında lstKitaplar_SelectedIndexChanged olay procedure'ünün çalışmasını sağlamış oluyoruz. Nevar ki böyle bir durumda sayfanın Page_Load olay procedürünün de tekrar çalışmasını engellemek yada başka bir deyişle bir kere çalışmasını garantilemek için **if (Page.IsPostBack==false)** kontrolünü Page_Load olay procedure'üne ekliyoruz.



Şekil 7. AutoPostBack özelliği

```
private void IstKitaplar_SelectedIndexChanged(object sender, System.EventArgs e)
{
    SqlCommand cmdKitapBul=new SqlCommand("Kitap Bul",conFriends); /* SqlCommand nesnemizi oluşturuyoruz ve
    commandString parametresine Stored Procedure'ün ismini yazıyoruz.*/
    cmdKitapBul.CommandType=CommandType.StoredProcedure; /* Bu kez SqlCommand'in bir Stored Procedure çalıştıracağına
    işaret ediyoruz.*/
    cmdKitapBul.Parameters.Add("@id",SqlDbType.Int); /* "Kitap Bul" isimli Stored Procedure'de yer alan @id isimli parametreyi
    komut nesnemize bildirmek için SqlCommand nesnemizin, Parameters koleksiyonuna ekliyoruz.*/
    cmdKitapBul.Parameters[0].Value=IstKitaplar.SelectedValue; /* listBox nesnesinde seçilen öğenin değerini (ki bu değer ID
    değeridir) SelectedValue özelliği ile alıyor ve SqlCommand nesnesinin 0 indexli parametresi olan @id SqlParameter nesnesine
    atıyoruz. Artık SqlCommand nesnemizi çalıştırdığımızda , @id paramteresinin değeri olarak seçili listBox öğesinin değeri gönderilecek
    ve bu değere göre çalışan Select sorgusunun döndürdüğü sonuçlar SqlDataReader nesnemize yüklenecek.*/
    SqlDataReader dr;
    conFriends.Open(); // Bağlantımız açılıyor.
    dr=cmdKitapBul.ExecuteReader(CommandBehavior.CloseConnection); // Komut çalıştırılıyor.
```

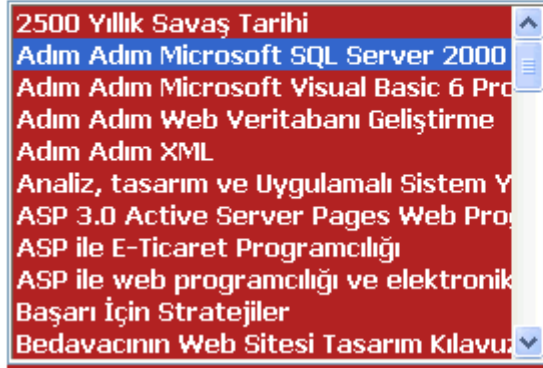


```

dgDetaylar.DataSource=dr; // dataGrid nesnesine veri kaynağı olarak SqlDataReader nesnemiz gösteriliyor.
dgDetaylar.DataBind(); // dataGrid verilere bağlanıyor.
dr.Close(); // SqlDataReader nesnemiz ve sonrada SqlConnection nesnemiz ( otomatik olarak ) kapatılıyor.
}

```

Kitap Listesi



ID	Kategori	Adı	Yazar	Basın
59	Bilgisayar Programlama	Adım Adım Microsoft SQL Server 2000 Programlama	Rebecca M. Riordan	ARK.

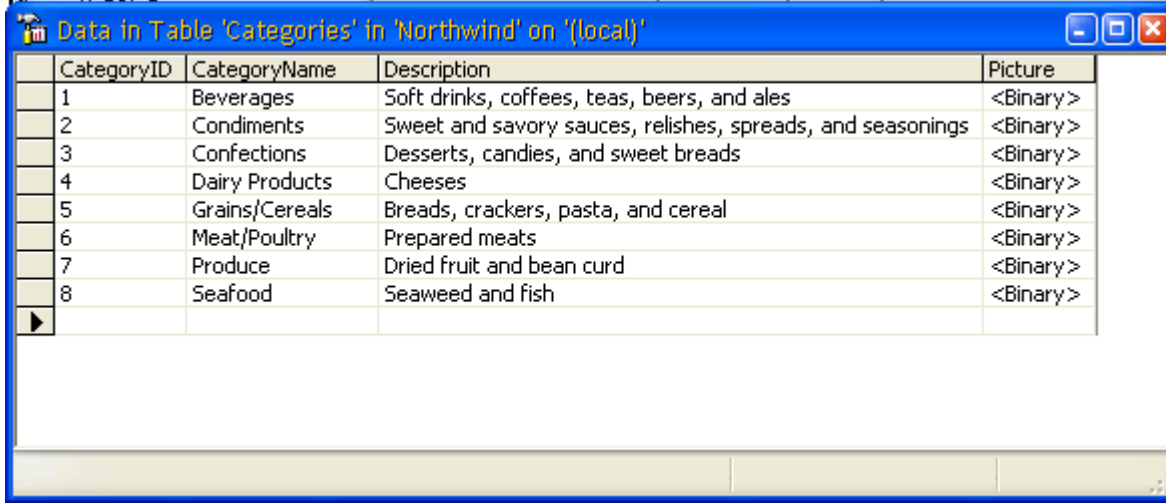
Şekil 8. Sonuç.

Geldik bir makalemizin daha sonuna. Yeni makalelerimizde görüşmek dileğiyle, hepinizi mutlu günler.

Stored Procedure Yardımıyla Tablodan Kayıt Silmek

Bugün ki makalemde Stored Procedure yardımıyla bir veritabanı tablosundan, bizim seçtiğimiz herhangi bir satırı nasıl sileceğimizi sizlere anlatmaya çalışacağım. Her zaman olduğu gibi örneğimizi geliştirmek için, SQL Server üzerinde yer alan Northwind veritabanını kullanmak istiyorum. SQL Server üzerinde çalışan örnekler geliştirmek istememin en büyük nedeni, bir veritabanı yönetim sistemi (Database Management System;DBMS) üzerinde .NET ile projeler geliştirmenin gerçekliği. Güncel yaşantımızda ağ üzerinde çalışan uygulamalar çoğunlukla , iyi bir veritabanı yönetim sistemi üzerinde yazılmış programlar ile gerçekleştirilmektedir. Çok katlı mimari olarak hepimizin kulağına bir şekilde gelmiş olan bu sistemde, aslında yazmış olduğumuz programlar, birer arayüz niteliği taşımakta olup kullanıcı ile veritabanı arasındaki iletişimi görsel anlamda kolaylaştıran birer araç haline gelmiştir. İşte bu sunum katmanı (presantation layer) denen yerdir. Burada veri tablolarını ve veritabanlarını üzerinde barındıran yer olarak veritabanı katmanı (Database Layer) büyük önem kazanmaktadır.

İşte bir önceki makalemde belirttiğim gibi Stored Procedure' leri kulanmamın en büyük amacı performans, hız ve güvenlik kriterlerinin önemidir. Dolayısıyla, örneklerimizi bu şekilde gerçek uygulamalara yakın tutarak, çalışırsak daha başarılı olacağımız inancındayım. Evet bu kadar laf kalabalığından sonra dilerseniz uygulamamıza geçelim. Uygulamamızın kolay ve anlaşılır olması amacıyla az satırlı bir tablo üzerinde işlemlerimizi yapmak istiyorum. Bu amaçla Categories tablosunu kullanacağım.



CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffees, teas, beers, and ales	<Binary>
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	<Binary>
3	Confections	Desserts, candies, and sweet breads	<Binary>
4	Dairy Products	Cheeses	<Binary>
5	Grains/Cereals	Breads, crackers, pasta, and cereal	<Binary>
6	Meat/Poultry	Prepared meats	<Binary>
7	Produce	Dried fruit and bean curd	<Binary>
8	Seafood	Seaweed and fish	<Binary>

Şekil 1. Categories tablosunda yer alan veriler.

Tablomuzun yapısını da kısaca incelersek ;

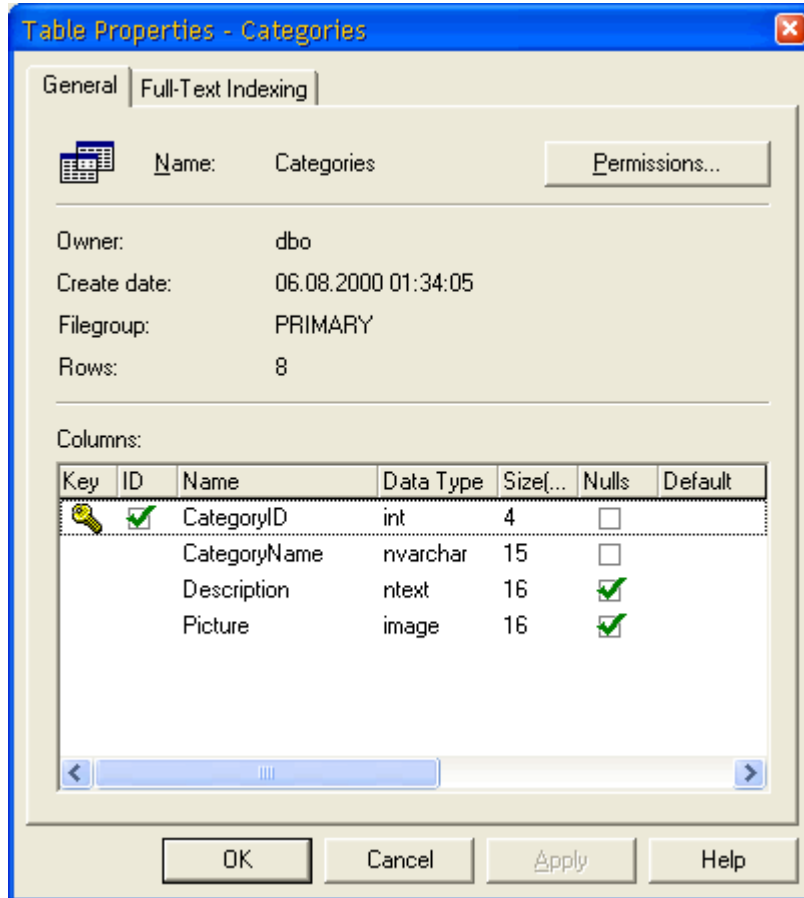
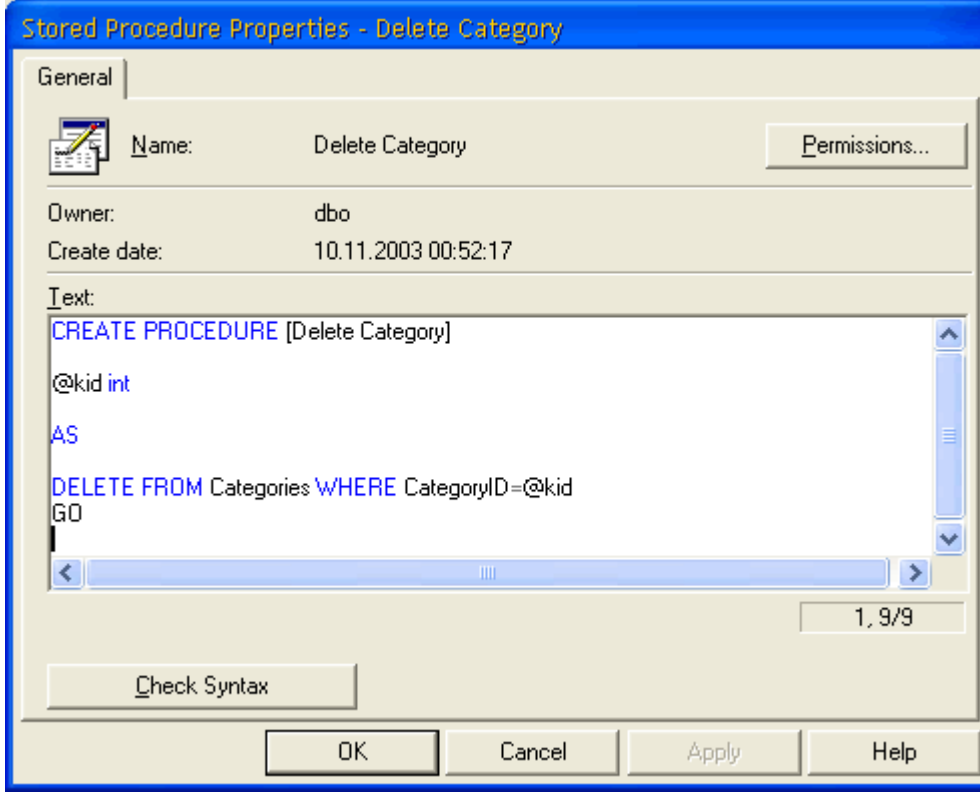


Table Properties - Categories						
General						
Name:		Categories		Permissions...		
Owner:	dbo					
Create date:	06.08.2000 01:34:05					
Filegroup:	PRIMARY					
Rows:	8					
Columns:						
Key	ID	Name	Data Type	Size(...)	Nulls	Default
	☒	CategoryID	int	4	☐	
		CategoryName	nvarchar	15	☐	
		Description	ntext	16	☒	
		Picture	image	16	☒	

Şekil 2. Categories tablosunun alan yapısı.

Burada CategoryID alanı bizim için önemlidir. Nitekim silme işlemi için kullanacağımız Stored Procedure içerisinde , belirleyici alan olarak bir parametreye dönüşecektir. Şimdi dilerseniz, Stored Procedure'ümüzü yazalım.

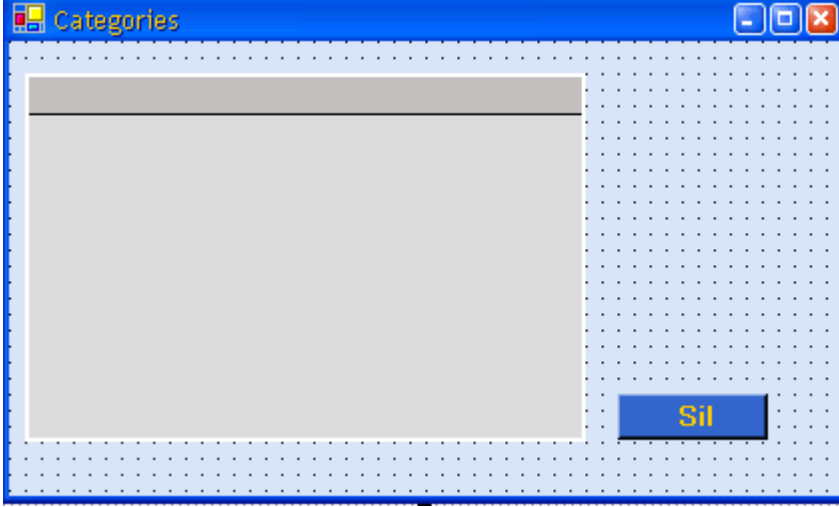


Şekil 3. Stored Procedure Kodları

```
CREATE PROCEDURE [Delete Category]
@kid int
AS
DELETE FROM Categories WHERE CategoryID=@kid
GO
```

Görüldüğü gibi burada son derece kolay bir T-SQL(Transact SQL) cümlecigi var. Burada yapılan işlem aslında @kid parametresine geçilen değeri CategoryID alanı ile eşleştirmek. Eğer bu parametre değerine karşılık gelen bir CategoryID değeri varsa; bu değeri taşıyan satır Categories isimli tablodan silinecektir.

Evet şimdi de .NET ortamında formumuzu tasarlayalım. New Project ile yeni bir C# projesi açarak işe başlıyoruz. Formumuzun tasarımını ben aşağıdaki şekilde yaptım. Sizde buna uygun bir form tasarlayabilir yada aynı tasarımı kullanabilirsiniz. Visual Studio.NET ile program geliştirmenin belkide en zevkli ve güzel yanı form tasarımları. Burada gerçekten de içimizdeki sanatçı ruhunu ortaya çıkartma imkanına sahibiz. Ve doğruyu söylemek gerekirse Microsoft firmasında artık içimizdeki sanatçı çocuğu özellikle bu tarz uygulamalarda, daha kolay açığa çıkartabilmemiz için elinden geleni yapıyor. Doğal olarak çok da güzel sonuçlar ortaya çıkıyor. Birde o eski bankalardaki (halen daha varya) siyah ekranlarda, incecik, kargacık, burgacık tasarımları ve arayüzleri düşünün. F12 ye bas geri dön. Tab yap. Şimdi F4 kodu gir.



Şekil 4. Formun İlk Yapısı

Formumuzda bir adet dataGrid nesnesi ve bir adetde button nesnesi yer alıyor. DataGrid nesnesini Categories tablosu içersinde yer alan bilgileri göstermek için kullanacağız. Datagrid verileri gösterirken kullanıcının kayıt eklemek, düzenlemek, ve seçtiği satırı buradan silmesini egellemek istediğimden ReadOnly özelliğine True değerini aktardım. Örneğimizin amacı gereği silme işlemini Sil textine sahip btnSil button nesnesinin Click olay procedure'ünden yapacağız. Elbette burada database'deki bilgileri dataGrid içersinde göstermek amacıyla bir SqlDataAdapter nesnesi kullanacağım. Bu sadece Categories isimli tablo içersindeki tüm satırları seçecek bir Select sorgusuna sahip olacak ve bunları dataGrid ile ilişkili olan bir DataTable nesnesine aktaracak.

Dilerseniz kodlarımızı yazmaya başlayalım. Öncelikle SqlConnection nesnemiz yardımıyla, Northwind veritabanına bir bağlantı açıyoruz. Daha sonra SqlDataAdapter nesnemizi oluşturuyoruz. SqlDataAdapter nesnesini yaratmak için new anahtar sözcüğü ile kullanabileceğimiz 4 adet overload constructor var. Overload constructor, aynı isme sahip yapıcı metodlar anlamına geliyor. Yani bir SqlDataAdapter nesnesini yaratabileceğimiz 4 kurucu(constructor) metod var ve bunların hepside aynı isme sahip(Overload;aşırı yüklenmiş) metodlar. Yeri gelmişken bunlardan da bahsederek bilgilerimizi hem tazeleyelim hem de arttırmış olalım. İşte bu yapıcı metodların prototipleri.

```
public SqlDataAdapter();
public SqlDataAdapter(string selectCommandText,string connectionString);
public SqlDataAdapter(string selectCommandText,SqlConnection selectConnection);
public SqlDataAdapter(SqlCommand selectCommand);
```

Ben uygulamamda ilk yapıcı metodu baz almak istiyorum. Evet artık kodlarımızı yazalım.

**NOT: Her zaman olduğu gibi projemizin başına
System.Data.SqlClient namespace ini eklemeyi unutmayalım.**

```
using System;
using System.Drawing;
using System.Collections;
```

```

using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;

/* Önce SqlConnection nesnesi yardimiyla SQL Server üzerinde yer alan, Northwind isimli veritabanina bir baglanti nesnesi
tanimliyoruz. İlk parametre yani initial catalog, veritabaninin ismini temsil ediyor. Şu anda SQL Server' in üzerinde yer alan
makinede çalıştığımızdan Data Source parametresine localhost degerini aktardik. */
SqlConnection conNorthwind=new SqlConnection("initial catalog=Northwind;Data Source=localhost;integrated
security=sspi;packet size=4096");

/* Şimdi Categories tablosunu bellekte temsil edecek olan DataTable nesnemizi yaratıyoruz. Dikkat edersek bellekte dedik.
DataTable nesnesi Categories tablosundaki verileri programin çalıştigi bilgisayar üzerindeki bellekte sakliyacaktir. Bu durumda
SqlConnection nesnemizin açık kalmasına gerek yoktur. Bu da elbetteki sunucu üzerindeki yükü azaltan bir etkidir.*/
DataTable dtbCategories=new DataTable("Kategoriler");
private void Form1_Load(object sender, System.EventArgs e)
{
    conNorthwind.Open();//Bağlantımızı açıyoruz.
    SqlDataAdapter da=new SqlDataAdapter();//Bir SqlDataAdapter nesnesi tanımladik.
    /* Asagidaki satir ile yarattigimiz SqlDataAdapter nesnesine bir Select sorgusu eklemis oluyoruz. Bu sorgu sonucu dönen deger
kumesi, SqlDataAdapter nesnesinin Fill metodunu kullandigimizda DataTable' in içerisinde hangi veriler ile dolduracagimizi belirtecek
önemli bir özelliktir. Bir SqlDataAdapter nesnesi yaratildiginda, SelectCommand özelligine SqlCommand türünden bir nesne
atanarak bu islem gerçekleştirilir. Burada aslında, SelectCommand özelliginin prototipinden dolayı new anahtar sözcüğü kullanılarak
bir SqlCommand nesnesi parametre olarak verilen select cümlecigi ile oluşturulmus ve SelectCommand özelligine atanmistir.
    *
    * public new SqlCommand SelectCommand
    * {
    * get;
    * set;
    * }
    * Prototipten de görüldüğü gibi SelectCommand özelliginin tipi SqlCommand nesnesi türündendir. Bu yüzden new
SqlCommand("....") ifadesi kullanilmistir.
    */
    da.SelectCommand=new SqlCommand("SELECT * FROM Categories");
    da.SelectCommand.Connection=conNorthwind; /* Select sorgusunun çalıştirilacagi baglanti belirlenir.*/
    da.FillSchema(dtbCategories,SchemaType.Mapped);/* Burada dataTable nesnemize, veritabanında yer alan Categories isimli
tablonun Schema bilgilerinide yüklüyoruz. Yani primaryKey bilgileri, alanların bilgileri yükleniyor. Bunu yapmamızın sebebi, Stored
Procedure ile veritabanındaki Categories tablosundan silme işlemini yapmadan önce , bellekteli tablodan da aynı satırı silip dataGrid
içindeki görüntünün ve DataTable nesnesinin güncel olarak kalmasını sağlamak. Nitekim silme işleminde DataTable nesnesinden
seçili satırı silmek için kullanacağımız Remove metodu PrimaryKey alanının deęerini istemektedir. Bunu verebilmek için tablonun
PrimaryKey bilgisininde belleęe yani bellekteki DataTable nesnesine yüklenmiş olması gerekir. İşte bu amaçla Schema bilgilerinide

```

alıyoruz*/

```

da.Fill(dtbCategories);/* Burada SqlDataAdapter nesnesinin Fill metodu çağırılır. Fill metodu öncelikle
SelectCommand.CommandText in degeri olan Select sorgusunu çalıştırır ve dönen veri kümesini dtbCategories isimli DataTable
nesnesinin bellekte referans ettiği alana yükler. Artık bağlantıyı da kapatabiliriz.*/
conNorthwind.Close();
/* Şimdi dataGrid nesnemize veri kaynağı olarak DataTable nesnemizi göstereceğiz. Böylece DataGrid, Categories tablosundaki
veriler ile dolucak. */
dgCategories.DataSource=dtbCategories;
}
private void btnDelete_Click(object sender, System.EventArgs e)
{
/* Şimdi silme işlemini gerçekleştireceğimiz Stored Procedure'e DataGrid nesnesi üzerinde kullanıcının seçmiş olduğu satırın
CategoryID sütununun değerini göndereceğiz. Bunun için kullanıcının seçtiği satırın numarasını CurrentCell.RowNumber özelliği ile
alıyoruz. Daha sonra, CategoryID sütunu dataGrid'in 0 indexli sütunu olduğundan CategoryID değerini elde ederken
dgCategories[currentRow,0] metodunu kullanıyoruz.*/
int currentRow;
int selectedCategoryID;
currentRow=dgCategories.CurrentCell.RowNumber;
selectedCategoryID=(int)dgCategories[currentRow,0]; /* Burada dgCategories[currentRow,0] aslında object tipinden bir değer
döndürür. Bu yüzden açık olarak dönüştürme dediğimiz (Explicit) bir Parse(dönüştürme) işlemi yapıyoruz. */
/* Şimdi de Stored Procedure'ümüzü çalıştıracak olan SqlCommand nesnesini tanımlayalım*/
SqlCommand cmdDelete=new SqlCommand();
cmdDelete.CommandText="Delete Category";/* Stored Procedure'ün adı CommandText özelliğine atanıyor. Ve bu stringin bir
Stored Procedure'e işaret ettiğini belirtmek için CommandType değerini CommandType.StoredProcedure olarak belirliyoruz.*/
cmdDelete.CommandType=CommandType.StoredProcedure;
cmdDelete.Connection=conNorthwind;//Komutun çalıştırılacağı bağlantı belirleniyor.
/* Şimdi ise @id isimli parametremizi oluşturacağız ve kullanıcının seçmiş olduğu satırın CategoryID değerini bu parametre ile
Stored Procedure'ümüze göndereceğiz.*/
cmdDelete.Parameters.Add("@kid",SqlDbType.Int);
cmdDelete.Parameters["@kid"].Value=selectedCategoryID;
/* Ve önemli bir nokta. Kullanıcıyı uymalıyız. Gerçekten seçtiği satırı silmek istiyor mu? Bunun için MessageBox nesnesini ve
Show metodunu kullanacağız. Bu metodun dönüş değerini DialogResult tipinde bir değişkenle kontrol ettiğimize dikkat edin.*/
DialogResult result;
result=MessageBox.Show("CategoryID : "+selectedCategoryID.ToString()+" . Bu satırı silmek istediğinizden emin
misiniz?", "Sil", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button1);
if (result==DialogResult.Yes ) /*Eğer kullanıcının cevabı evet ise aşağıdaki kod bloğundaki kodlar çalıştırılır ve satır önce
DataTable nesnesinden sonrada kalıcı olarak databaseden silinir.*/
{
conNorthwind.Open();// Bağlantımızı açıyoruz.

```

/* Elbette veritabanından doğrudan sildiğimiz satırı bellekteki DataTable nesnesinin referans ettiği yerdende siliyoruz ki datagrid nesnemiz güncelliğini korusun. Bunun için seçili olan dataTable satırını bir DataRow nesnesine aktarıyoruz. Bunu yaparkende seçili kaydı Find metodu ile CategoryID isimli Primary Key alanı üzerinden arama yapıyoruz. Kayıt bulunduğunda tüm satırbilgisi bir DataRow türü olarak geri dönüyor ve bunu DataRow nesnemize atıyoruz. Remove metodu silinmek istenen satır bilgisini parameter olarak alır. Ve bu parameter DataRow tipinden bir parametredir.*/

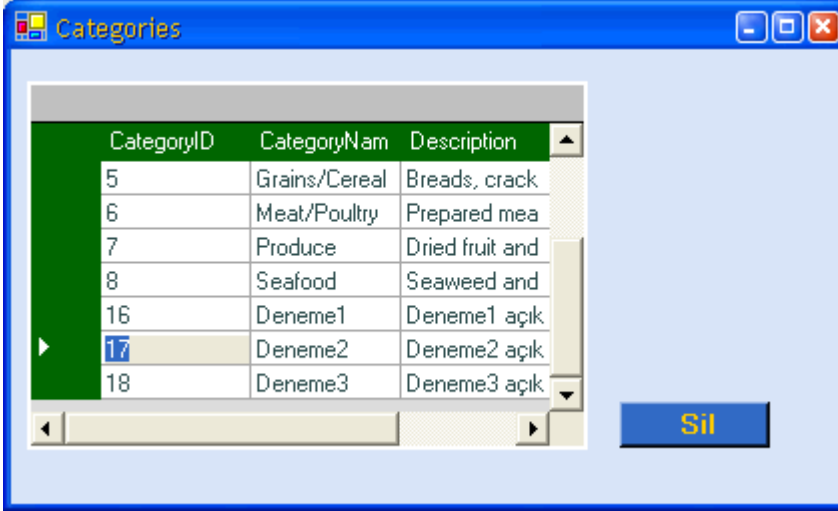
```
DataRow drSelectedRow;
drSelectedRow=dtbCategories.Rows.Find(selectedCategoryID);
dtbCategories.Rows.Remove(drSelectedRow);
cmdDelete.ExecuteNonQuery();/ * Artık Stored Procedure de çalıştırılıyor ve slime işlemi doğrudan veritabanındaki tablo üzerinden gerçekleştiriliyor. ExecuteNonQuery bu Stored Procedure'ü çalıştıracak olan metoddur. Delete,Update,Insert gibi kayıt döndürmesi beklenmeyen (Select sorguları gibi) sql cümlecikleri için ExecuteNonQuery metodu kullanılır.*/
conNorthwind.Close();
}
}
```

Şimdi dilerseniz programımızı çalıştırıp sonuçlarına bir bakalım. Öncelikle Categories isimli tabloya doğrudan SQL Server üzerinden örnek olması açısından bir kaç kayıt ekleyelim.

CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffees, teas, beers, and ales	<Binary>
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	<Binary>
3	Confections	Desserts, candies, and sweet breads	<Binary>
4	Dairy Products	Cheeses	<Binary>
5	Grains/Cereals	Breads, crackers, pasta, and cereal	<Binary>
6	Meat/Poultry	Prepared meats	<Binary>
7	Produce	Dried fruit and bean curd	<Binary>
8	Seafood	Seaweed and fish	<Binary>
16	Deneme1	Deneme1 açıklaması.	<Binary>
17	Deneme2	Deneme2 açıklaması.	<Binary>
18	Deneme3	Deneme3 açıklaması	<Binary>
*			

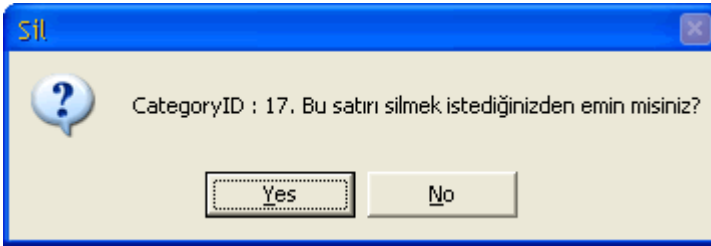
Şekil 5. Categories tablosuna 3 yeni kayıt ekledik.

Şimdi uygulamamızı çalıştıralım. Bu durumda ekran görüntüsü aşağıdaki gibi olacaktır. Şu anda dataGrid içindeki bilgiler veritabanından alınıp , bellekteki dataTable nesnesinin referans ettiği bölgedeki verilerden oluşmaktadır. Dolayısıyla Sql Server'a olan bağlantımız açık olmadığı halde verileri izleyebilmekteyiz. Hatta bunların üzerinde değişiklikler yapıp normal tablo işlemlerinde (silme,kayıt ekleme,güncelleme vb... gibi) gerçekleştirebiliriz. Bu bağlantısız katman olarak adlandırdığımız olaydır. Bu konuya ilerleyen makalelerimizde daha detaylı olarak inceleyeceğiz.



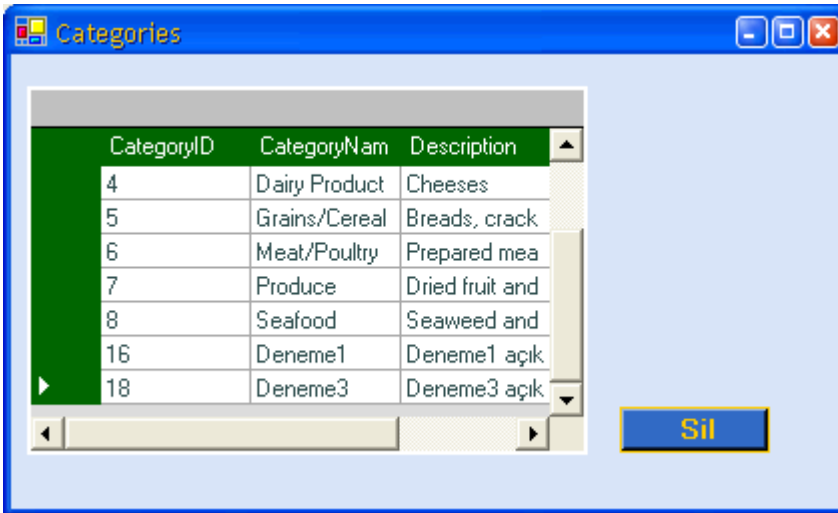
Şekil 6. Load Procedure'ünün çalıştırılmasından sonraki görünüm.

Şimdi seçtiğimiz 17 CategoryID satırını silelim. Ekranı bir soru çıkacaktır.



Şekil 7. MessageBox.Show(.....) metodunun sonucu.

Şimdi Yes butonuna basalım. Bu durumda 17 CategoryID li satır dataTable'dan dolayısıyla dataGrid'den silinir. Aynı zamanda çalıştırdığımız Stored Procedure ile veritabanından da doğrudan silinmiştir.



Şekil 8. Silme işlemi sonrası.

Şimdi SQL Server'a geri dönüp tablonun içeriğini kontrol edecek olursak aşağıdaki sonucu elde ederiz.

CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffee	<Binary>
2	Condiments	Sweet and savory	<Binary>
3	Confections	Desserts, candies	<Binary>
4	Dairy Products	Cheeses	<Binary>
5	Grains/Cereals	Breads, crackers	<Binary>
6	Meat/Poultry	Prepared meats	<Binary>
7	Produce	Dried fruit and beans	<Binary>
8	Seafood	Seaweed and fish	<Binary>
16	Deneme1	Deneme1 açıklaması	<Binary>
18	Deneme3	Deneme3 açıklaması	<Binary>

Şekil 9. Sonuç.

Görüldüğü gibi CategoryID=17 olan satır veritabanındaki tablodanda silinmiştir. Bir sonraki makalemizde görüşmek dileğiyle.

Overload Metodların Gücü

Değerli Okurlarım, Merhabalar.

Bu makalemde sizlere overload kavramından bahsetmek istiyorum. Konunun daha iyi anlaşılabilmesi açısından, ilerleyen kısımlarda basit bir örnek üzerinde de çalışacağız.

Öncelikle Overload ne demek bundan bahsedelim. Overload kelime anlamı olarak Aşırı Yükleme anlamına gelmektedir. C# programlama dilinde overload dendiğinde, aynı isme sahip birden fazla metod akla gelir. Bu metodlar aynı isimde olmalarına rağmen, farklı imzalara sahiptirler. Bu metodların imzalarını belirleyen unsurlar, parametre sayıları ve parametre tipleridir. Overload edilmiş metodları kullandığımız sınıflarda, bu sınıflara ait nesne örnekleri için aynı isme sahip fakat farklı görevleri yerine getirebilen (veya aynı görevi farklı sayı veya tipte parametre ile yerine getirebilen) fonksiyonellikler kazanmış oluruz.

Örneğin;

```
public string MetodA(int a)
```

```
public int MetodA(int a,int c)
```

```
public void MetodA(string d)
```

Şekil 1 : Overload metodlar.

Şekil 1 de MetodA isminde 3 adet metod tanımı görüyoruz. Bu metodlar aynı isme sahip olmasına rağmen imzaları nedeni ile birbirlerinden tamamiyle farklı metodlar olarak algılanırlar. Bize sağladığı avantaj ise, bu metodları barındıran bir sınıf nesnesi yarattığımızda aynı isme sahip metodları farklı parametreler ile çağırabilmemizdir. Bu bir anlamda her metoda farklı isim vermek gibi bir karışıklığında bir nebze önüne geçer. Peki imza dediğimiz olay nedir? Bir metodun imzası şu unsurlardan oluşur.

Metod İmzası Kabul Edilen Unsurlar	Metod İmzası Kabul Edilmeyen Unsurlar
Parametre Sayısı Parametrenin Tipleri	Metodun Geri Dönüş Tipi

Tablo 1. Kullanım Kuralları

Yukarıdaki unsurlara dikkat ettiğimiz sürece dilediğimiz sayıda aşırı yüklenmiş (overload edilmiş) metod yazabiliriz. Şimdi dilerseniz küçük bir Console uygulaması ile , overload metod oluşumuna engel teşkil eden duruma bir göz atalım.Öncelikle metodun geri dönüş tipinin metodun imzası olarak kabul edilemeyeceğinden bahsediyoruz. Aşağıdaki örneğimizi inceleyelim.

```
using System;
namespace Overloading1
{
    class Class1
    {
        public int Islem(int a)
        {
            return a*a;
        }
        public string Islem(int a)
        {
            string b=System.Convert.ToString(a);
            return "Yaşım:"+b;
        }
        [STAThread]
        static void Main(string[] args)
        {
        }
    }
}
```

Örneğin yukarıdaki uygulamada, Islem isimli iki metod tanımlanmıştır. Aynı parametre tipi ve sayısına sahip olan bu metodların geri dönüş değerlerinin farklı olması nedeni ile derleyici tarafından farklı metodlar olarak algılanmış olması gerektiği düşünülebilir. Ancak böyle olmamaktadır. Uygulamayı derlemeye çalıştığımızda aşağıdaki hata mesajı ile karşılaşırız.

Overloading1.Class1' already defines a member called 'Islem' with the same parameter types

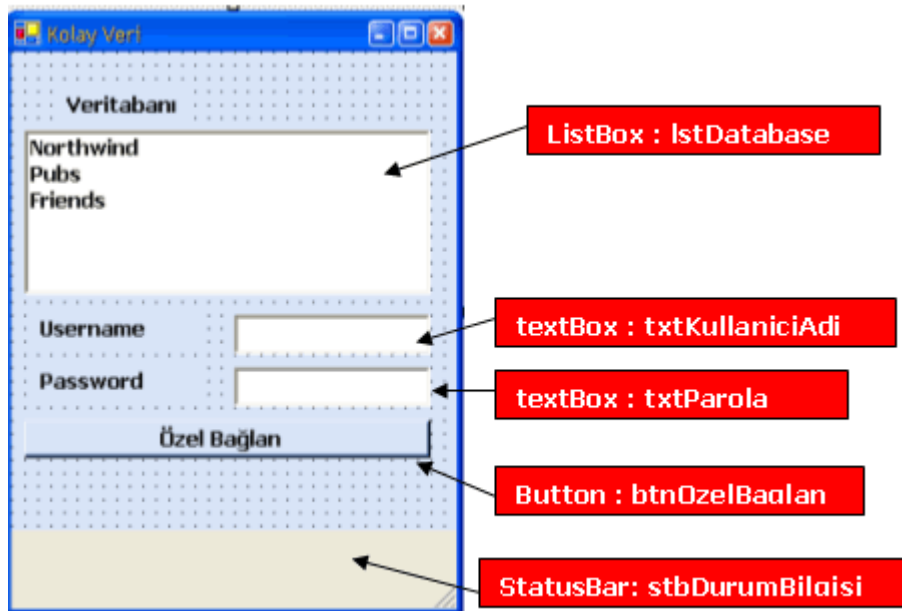
Yapıcı metodlarda overload edebiliriz. Bu da oldukça önemli bir noktadır. Bunu zaten .NET ile program geliştirirken sıkça kullanırız. Örneğin SqlConnection sınıfından bir nesne örneği yaratmak istediğimizde, bunu yapabileceğimiz 2 overload edilmiş yapıcı metod olduğunu görürüz. Bunlardan birisi aşağıda görünmektedir.



Şekil 2. Örnek bir Overload Constructor(Aşırı Yüklenmiş Yapıcı) metod.

Dolayısıyla bizde yazdığımız sınıflara ait constructorları overload edebiliriz. Şimdi dilerseniz overload ile ilgili olarak kısa bir uygulama geliştirelim. Bu uygulamada yazdığımız bir sınıfa ait constructor metodları overload ederek değişik tipte fonksiyonellikler edinmeye çalışacağız.

Bu uygulamada KolayVeri isminde bir sınıfımız olucak. Bu sınıfın üç adet yapıcısı olucak. Yani iki adet overload constructor yazacağız. İki tane diyorum çünkü C# zaten default constructoru biz yazmasak bile uygulamaya ekliyor. Bu default constructorlar parametre almayan constructorlardır. Overload ettiğimiz constructor metodlardan birisi ile, seçtiğimiz bir veritabanına bağlanıyoruz. Diğer overload metod ise, parametre olarak veritabanı adından başka, veritabanına bağlanmak için kullanıcı adı ve parola parametrelerini de alıyor. Nitekim çoğu zaman veritabanlarımızda yer alan bazı tablolara erişim yetkisi sınırlamaları ile karşılaşabiliriz. Bu durumda bu tablolara bağlantı açabilmek için yetkili kullanıcı adı ve parolayı kullanmamız gerekir. Böyle bir olayı canlandırmaya çalıştım. Elbetteki asıl amacımız overload constructor metodların nasıl yazıldığını, nasıl kullanıldığını göstermek. Örnek gelişmeye çok, hemde çok açık. Şimdi uygulamamızın bu ilk kısmına bir gözatalım. Aşağıdakine benzer bir form tasarımı yapalım.



Şimdi sıra geldi kodlarımızı yazmaya. Öncelikle uygulamamıza KolayVeri adında bir class ekliyoruz. Bu class'ın kodları aşağıdaki gibidir. Aslında uygulamaya bu aşamada baktığımızda SqlConnection nesneminin bir bağlantı oluşturmasını özelleştirmiş gibi oluyoruz. Gerçektende aynı işlemleri zaten SqlConnection nesnesini overload constructor'lari ile yapabiliyoruz. Ancak temel

amacımız aşırı yüklemeyi anlamak olduğu için programın çalışma amacının çok önemli olmadığı düşüncesindeyim. Umuyorum ki sizlere aşırı yükleme hakkında bilgi verebiliyor ve vizyonunuzu geliştirebiliyorumdur.

```
using System;
using System.Data.SqlClient;
namespace Overloading
{
    public class KolayVeri
    {
        private string baglantiDurumu; /* Connection'ın durumunu tutacak ve sadece bu class içinde geçerli olan bir string değişken tanımladık. private anahtar kelimesi değişkenin sadece bu class içerisinde yaşayabileceğini belirtir. Yazmayabiliriz, nitekim C# default olarak değişkenleri private kabul eder.*/

        public string BaglantiDurumu /* Yukarıda belirttiğimiz baglantiDurumu isimli değişkenin sahip olduğu değeri, bu class'a ait nesne örneklerini kullandığımız yerde görebilmek için sadece okunabilir olan (readonly), bu sebeplede sadece Get bloğuna sahip olan bir özellik tanımlıyoruz.*/
        {
            get
            {
                return baglantiDurumu; /* Bu özelliğe eriştiğimizde baglantiDurumu değişkeninin o anki değeri geri döndürülecek. Yani özelliğin çağırıldığı yere döndürülecek.*/
            }
        }

        public KolayVeri() /* İşte C# derleyicisinin otomatik olarak eklediği parametresiz yapıcı metod. Biz bu yapıcıya tek satırlık bir kod ekliyoruz. Eğer nesne örneği parametresiz bir Constructor ile yapılırsa bu durumda bağlantının kapalı olduğunu belirtmek için baglantiDurumu değişkenine bir değer atıyoruz. Bu durumda uygulamamızda bu nesne örneğinin BaglantiDurumu özelliğine eriştiğimizde BAGLANAMADIK değerini elde edeceğiz.*/
        {
            baglantiDurumu="BAGLANAMADIK";
        }

        public KolayVeri(string veritabaniAdi) /* Bizim yazdığımızı aşırı yüklenmiş ilk yapıcı metoda gelince. Burada yapıcımız, parametre olarak bir string alıyor. Bu string veritabanının adını barındıracak ve SqlConnection nesnemiz için gerekli bağlantı stringine bu veritabanının adını geçirecek.*/
        {
            string connectionString="initial catalog="+veritabaniAdi+";data source=localhost;integrated security=sspi";
            SqlConnection con=new SqlConnection(connectionString); /* SqlConnection bağlantımız yaratılıyor.*/
            try /* Bağlantı işlemini bir try bloğunda yapıyoruz ki, herhangi bir nedenle Sql sunucusuna bağlantı sağlanamassa (örneğin hatalı veritabanı adı nedeni ile) catch bloğunda baglantiDurumu değişkenine BAGLANAMADIK değerini atıyoruz. Bu durumda program içinde KolayVeri sınıfından örnek nesnenin BaglantiDurumu özelliğinin değerine baktığımızda BAGLANAMADIK değerini alıyoruz böylece bağlantının sağlanamadığına kanaat getiriyoruz. Kanaat dedikte aklıma Üsküdar'da ki Kanaat lokantası
```

geldi :) Yemekleri çok güzeldir. Sanırım karnımız acıktı değerli okuyucularım.Neyse kaldığımız yerden devam edelim.*/

```
{
    con.Open(); // Bağlantımız açılıyor.
    baglantiDurumu="BAGLANDIK"; /* BaglantiDurumu özelliğimiz (Property), baglantiDurumu değişkeni sayesinde
BAGLANDIK değerini alıyor.*/
}
catch(Exception hata) /* Eğer bir hata olursa baglantiDurumu değişkenine BAGLANAMADIK değerini atıyoruz.*/
{
    baglantiDurumu="BAGLANAMADIK";
}
}

public KolayVeri(string veritabaniAdi,string kullanıcıAdi,string parola) /* Sıra geldi ikinci overload constructor metoda. Bu
metod ekstradan iki parametre daha alıyor. Bir tanesi user id ye tekabül edecek olan kullanıcıAdi, diğeri ise bu kullanıcı için
password'e tekabül edecek olan parola. Bunlari SqlConnection'in connection stringine alarak , veritabanına belirtilen kullanıcı ile giriş
yapmış oluyoruz. Kodların işleyişi bir önceki metodumuz ile aynı.*/
{
    string connectionString="initial catalog="+veritabaniAdi+";data source=localhost;user
id="+kullanıcıAdi+";password="+parola;
    SqlConnection con=new SqlConnection(connectionString);
    try
    {
        con.Open();
        baglantiDurumu="BAGLANDIK";
    }
    catch(Exception hata)
    {
        baglantiDurumu="BAGLANAMADIK";
    }
}
}
```

Şimdi sıra geldi, formumuz üzerindeki kodları yazmaya.

```
string veritabaniAdi;
private void lstDatabase_SelectedIndexChanged(object sender, System.EventArgs e)
{
    veritabaniAdi=lstDatabase.SelectedItem.ToString();
    /* Burada kv adında bir KolayVeri sınıfından nesne örneği (object instance) yaratılıyor. Dikkat edecek olursanız burada yazdığımı
ikinci overload constructor'u kullandık.*/
```

KolayVeri kv=new KolayVeri(veritabaniAdi); /* Burada KolayVeri(dediğimizde .NET bize kullanabileceğimiz aşırı yüklenmiş constructorları aşağıdaki şekilde olduğu gibi hatırlatacaktır. IntelliSense'in gözünü seveyim.*/

```
private void lstDatabase_SelectedIndexChanged(object sender, System.EventArgs e)
{
    string veritabaniAdi=lstDatabase.SelectedItem.ToString();
    KolayVeri kv=new KolayVeri(
    stbDurumBilgi 2 of 3 KolayVeri.KolayVeri(string veritabaniAdi)String()+" "+kv.BaglantiDurumu;

    KolayVeri kvOzel=new KolayVeri(
}
```

stbDurumBilgisi.Text=lstDatabase.SelectedItem.ToString()+" "+kv.BaglantiDurumu;

private void btnOzelBaglan_Click(object sender, System.EventArgs e)

{

string kullanıcı,sifre;

kullanıcı=txtKullaniciAdi.Text;

sifre=txtParola.Text;

veritabaniAdi=lstDatabase.SelectedItem.ToString();

KolayVeri kvOzel=new KolayVeri(veritabaniAdi,kullanıcı,sifre); /* Burada ise diğer aşırı yüklenmiş yapıcıımızı kullanarak bir KolayVeri nesne örneği oluşturuyoruz.*/

```
private void lstDatabase_SelectedIndexChanged(object sender, System.EventArgs e)
{
    string veritabaniAdi=lstDatabase.SelectedItem.ToString();
    KolayVeri kv=new KolayVeri(veritabaniAdi);
    stbDurumBilgisi.Text=lstDatabase.SelectedItem.ToString()+" "+kv.BaglantiDurumu;

    KolayVeri kvOzel=new KolayVeri(
    3 of 3 KolayVeri.KolayVeri(string veritabaniAdi, string kullanıcıAdi, string parola)
```

stbDurumBilgisi.Text=lstDatabase.SelectedItem.ToString()+" "+kvOzel.BaglantiDurumu+" User:"+kullanıcı;

}

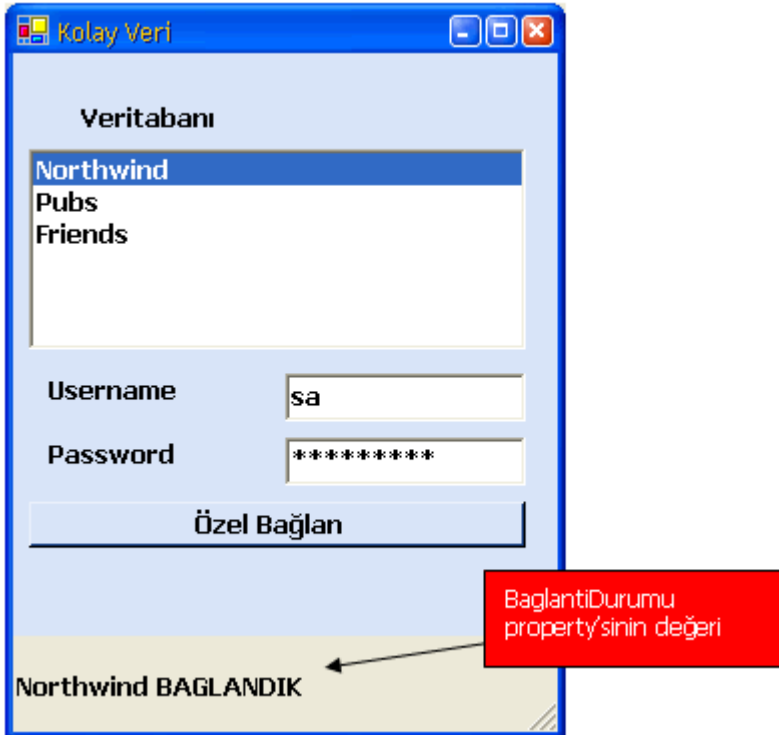
}

Evet şimdide programın nasıl çalıştığına bir bakalım. Listbox nesnesi üzerinde bir veritabanı adına bastığımızda bu veritabanına bir bağlantı açılır.



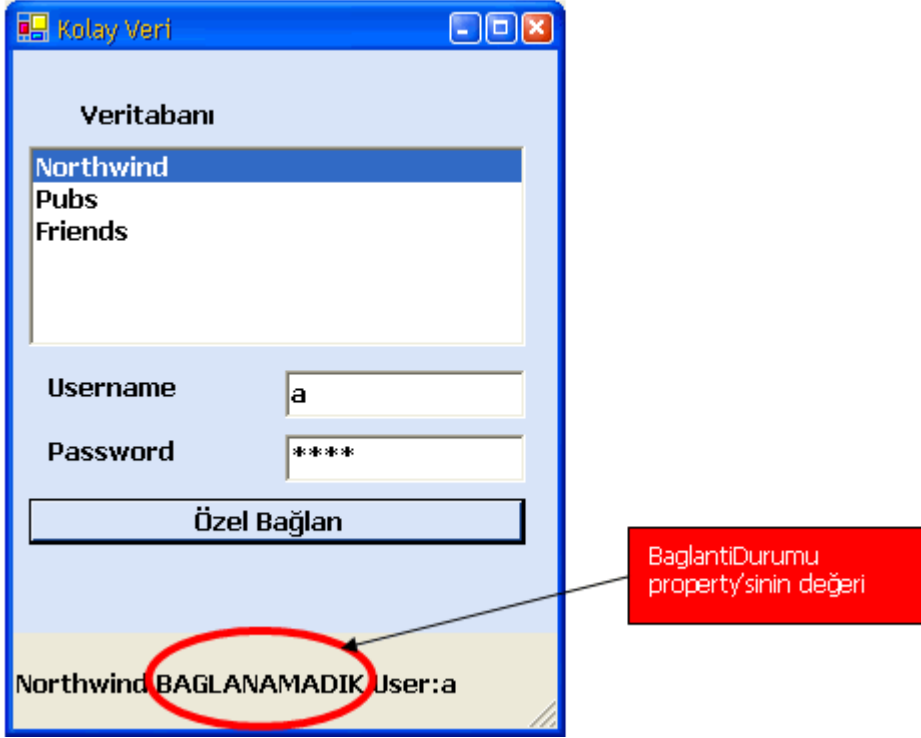
Şekil 6. Listboxta tıklanan veritabanına bağlandıktan sonra.

Ve birde kullanıcı adı ile parola verilerek nasıl bağlanacağımızı görelim.



Şekil 7. Kullanıcı adı ve parola ile bağlantı

Peki ya yanlış kullanıcı adı veya parola girersek.



Şekil 8. Yanlış kullanıcı adı veya parolası sonrası.

Evet değerli MsAkademik okuyucuları bu seferlikte bu kadar. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler, yarınlar dilerim.

Transaction Kavramı

Değerli Okurlarım, Merhabalar.

Bu makalemizde sizlere veritabanı programcılığında ve özellikle de çok katlı mimaride çok önemli bir yere sahip olan Transaction'lar hakkında bilgi vermeye çalışacağım. Her zaman olduğu gibi konuyu iyi anlayabilmek için bir de örnek uygulamamız olacak. Öncelikle Transaction nedir, ne işe yarar bunlardan bahsedelim. Çoğu zaman programlarımızda ardı arkasına veritabanı işlemleri uygulatırız. Örneğin, bir veritabanındaki bir tablodan kayıt silerken, aynı olayın sonucunda başka bir ilişkili tabloya silinen bu verileri ekleyebilir veya güncelleyebiliriz. Hatta bu işlemin arkasından da silinen kayıtların bulunduğu tablo ile ilişkili başka tablolaradan da aynı verileri sildiğimiz işlemleri başlatabiliriz. Dikkat edecek olursanız burada birbirleriyle ilintili ve ardışık işlemlerden söz ediyoruz.

Farzedelim ki, üzerinde çalıştığımız bu tablolara farklı veritabanı sunucularında bulunsun. Örneğin, birisi Adana'da diğeri Arnavutluk'ta ortağı olduğumuz şirketin sunucularında. Hatta bir diğeri Kazakistandaki ortağımızın bir kaç sunucusunda bulunuyor olsun. E hadi bir tanede bizim sunucumuzda farklı bir veya bir kaç tablo olsun. Şimdi düşünün ki, biz Kazakistan'a sattığımız malların bilgisini, Arnavutluk'taki ortağımızın sunucularınada bildiriyoruz. Stoğumuzda bulunan mallarda 1000 adet televizyonu Kazakistana göndermek amacıyla ihrac birimimize rapor ediyoruz. İhrac birimi ilgili işlemleri yaptıktan sonra, Kazakistandaki sunuculardan gelen ödeme bilgisini alıyor. Sonra ise stok tan' 1000 televizyonu düşüyor, muhasebe kayıtlarını

güncelliyor, Kazakistan’ daki sunucularda stok artışını ve hesap eksilişlerini bildiriyor. Daha sonra ise , Arnavutluk’ taki sunuculara stok artışı ve stok azalışlarını ve muhasebe hareketlerini belirtiyor. Senaryo bu ya. Çok hızlı bir teknolojik alt yapıya sahip olduğumuzu ve bankalardan şirkete olan para akışlarının anında görülüp , tablolara yansıtılabildiğini ve bu sayede de stoklardaki hareketlerin ve şirket muhasebe kayıtlarındaki hareketlerin hemen gerçekleşebileceğini yani mümkün olduğunu düşünelim. (Değerli okuyucalarım biliyorum ki uzun bir cümle oldu ama umarım gelmek istediğim noktayı anlamaya başlamışsınızdır)

Bahsettiğimiz tüm bu işlemler birer iş parçacığıdır ve aslında hepsi toplu olarak tek bir amaca hizmet etmektedir. Tüm sunucuları stok hareketlerinden ve gerekli muhasebe değişikliklerinden eş zamanlı olarak (aşağı yukarı yani) haberdar etmek ve veritabanı sunucularını güncellemek. Dolayısıyla tüm bu iş parçacıklarını, tek bir bütün işi gerçekleştirmeye çalışan unsurlar olduğunu söyleyebiliriz. İşte burada tüm bu iş parçacıkları için söylenebilecek bazı hususlar vardır. Öncelikle,

- İş parçacıklarının birinde meydana gelen aksaklık , diğer işlerin ve özellikle takip eden iş parçacıklarının doğru şekilde işlememesine neden olabilir. Dolayısıyla tüm bu iş parçacıkları başarılı olduğu takdirde bütün iş başarılı olmuş sayılabilir.
- Diğer yandan iş parçacıklarının işleyişi sırasında veriler üzerindeki değişikliklerin de tutarlı olması birbirlerini tamamlayıcı nitelik taşıması gerekir. Söz gelimi stoklarımızda 2000 televizyon varken 1000 televizyon ihraç ettiğimizde stoğumuza mal eklenmediğini düşünecek olursak 1000 televizyon kalması gerekir. 1001 televizyon veya 999 televizyon değil. İşte bu verilerin tutarlılığını gerektirir.

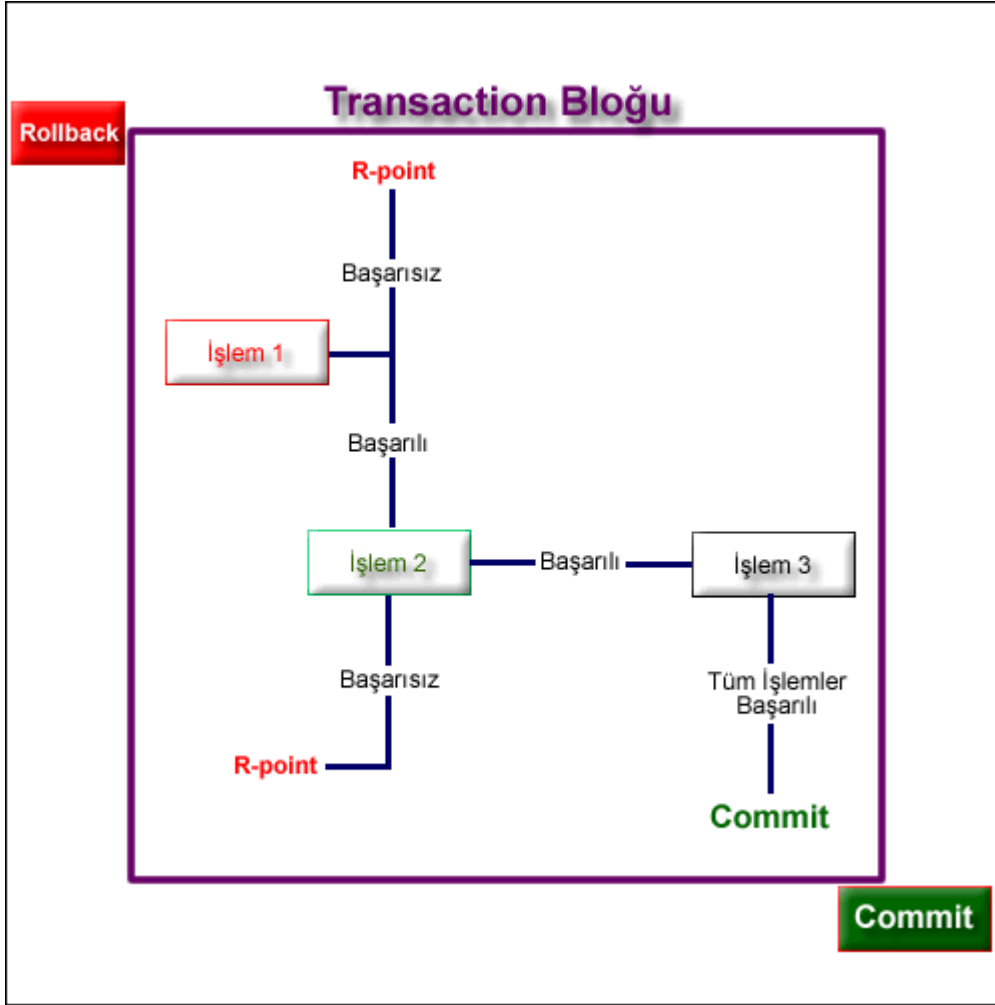
İşte bu nedenlerle ötürü Transaction kavramı ortaya çıkarmıştır. Bu kavrama göre aslında bahsedilen tüm iş parçakları kusursuz olarak başarılı olduklarında “işlem tamam” denebilir. İşte bizde veritabanı uygulamalarımızı geliştirirken, bu tip iş parçacıklarını bir Transaction bloğuna alırız. Şimdi ise karşımıza iki yeni kavram çıkacaktır. Commit ve Rollback.Eğer Transaction bloğuna dahil edilen iş parçacıklarının tümü başarılı olmuş ise Transaction Commit edilir ve iş parçacıklarındaki tüm veri değişimleri gerçekten veritabanlarına yansıtılır. Ama iş parçacıklarından her hangibirinde tek bir hata oluşup iş parçacığının işleyişi bozulur ise bu durumda tüm Transaction Rollback edilir ve bu durumda, o ana kadar işleyen tüm iş parçacıklarındaki işlemler geri alınarak , veritabanları Transaction başlamadan önceki haline döndürülür. Bu bir anlamda güvenlik ve verileri koruma adına oluşturulmuş bir koruma mekanizmasıdır.

Peki ya iş parçacıklarının düzgün işlemiyişine sebep olarak neler gösterebiliriz. Tabiki çevresel faktörler en büyük etkindir. Sunucuları birbirine bağlayan hatlar üzerinde olabilecek fiziki bir hasar işlemleri yarıda bırakabilir ve Kazakistan’ daki sunuculardaki 1000 televizyonluk artış buraya hiç yansımaz. Kazakistan’daki yetkili Türkiye’deki merkezi arayıp “stoğumda hala 1000 televizyon görünmüyor.” diyebilir. Merkezdeki yetkili ise. “Bizim stoklardan 1000 tv dün çıkmış. Bilgisayar kayıtları yalan mı söyleyecek kardeşim.” diyebilir. Neyseki Transactionlar sayesinde olay şöyle gelişir.

Kazakistan Büro : Stokta bir hareket yok bir sorunmu var acaba?

Merkez: Evet . Karadenizden geçen boru hattında fırtına nedeni ile kopma olmuş. Mallar bizim stokta halen daha çıkmadılar. Gemide bekletiyoruz.

Çok abartı bir senaryo oldu aslında. Nitekim o televizyonlar bir şekilde yerine ulaşır Transaction’lara gerek kalmadan. Ama olayı umarım size betimleyebilmişimdir. Şimdi gelin olayın teknik kısmını bir de grafik üzerinde görelim.



Şekil 1. Transaction Kavramı

Şekilde görüldüğü gibi örnek olarak 3 adet işlem parçacığı içeren bir Transaction bloğumuz var. Bu işlemler birbirine bağlı olarak tasvir edilmiştir. Eğer herhangi biri başarısız olursa veriler üzerinde o ana kadar olan değişiklikler geri alınır ve sistem Transaction başlamadan önceki haline konumlandırılır. Şekilimize bunlar R-point yani Rollback Noktasına git olarak tasvir edilmiştir. Ancak tüm işlemler başarılı olursa Transaction içinde gerçekleşen tüm veri değişiklikleri onaylanmış demektir.

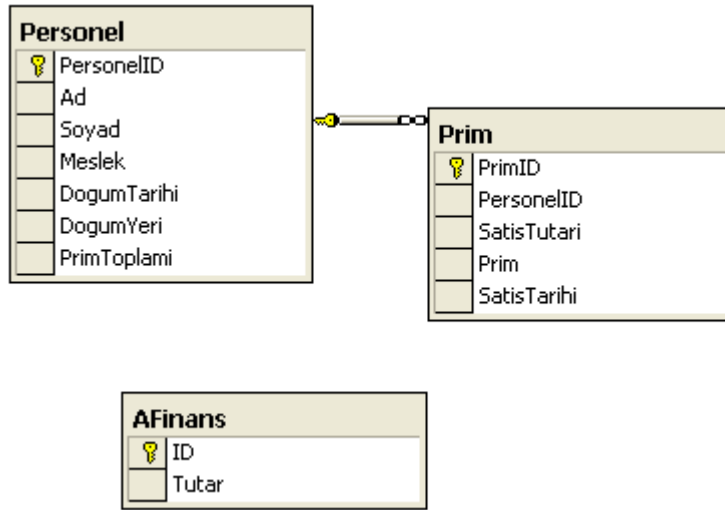
Transactionlar ile ilgili olarak önemli bir konu ise yukarıdaki örneklerde anlattığımız gibi birden fazla veritabanı olması durumunda bu Transaction işlemlerinin nasıl koordine edilceğidir. Burada Dağıtık Transaction dediğimiz Distributed Transaction kavramı ortaya çıkar. Bu konuyu ilerleyen makalelerimizde işlemey çalışacağım. Şimdilik sadece tek bir veritabanı üzerinde yazabileceğimiz Transaction' lardan bahsetmek istiyorum..NET içerisinde SqlConnection sınıfında yer alan nesneleri Transaction nesneleri kullanılarak bu işlemi gerçekleştirebiliriz. Ben SqlTransaction nesnesini ele alacağım. Bu nesneyi oluşturmak için herhangi bir yapıcı metod yoktur. SqlDataReader sınıfında olduğu gibi bu sınıfa ait nesneler birer değişkenmiş gibi tanımlanır. Nesne atamaları SqlConnection nesnesi ile gerçekleştirilir ve bu aynı zamanda Transaction'ın hangi SqlConnection bağlantısı için başlatılacağını belirlemeye yarar.

```
SqlTransaction tran;
tran = conNorthwind.BeginTransaction();
```

Yukarıdaki ifadeye dikkat edersek, bir SqlTransaction nesnesi tanımlanmış ve daha sonra conNorthwind isimli SqlConnection nesnesi için başlatılmıştır. İşte Transaction bloğunun başladığı nokta burasıdır. Şimdi ise , hangi Sql komutlarını dolayısıyla hangi iş parçacıklarını bu transaction nesnesine(yani bloğuna) dahil edeceğimizi belirlemeliyiz. Bu işlem genelde çalıştırılacak olan SqlCommand nesnelerinin Transaction özelliklerine Transaction nesnesinin atanması ile gerçekleştirilir. Dilerseniz gerçekçi bir örnek üzerinde çalışalım ve Transaction kavramını daha iyi anlayalım.

Merkezi İstanbul'da olan uluslararası devre mülk satan bir şirket, ülke ofislerinde satışını yaptığı devre mülkler için, satışı yapan personele ait banka hesaplarına EFT işlemi içeren bir uygulamaya sahip olsun. Bahsi geçen uygulamanın çalışmasına bir örnek verelim; Brezilya'daki ofisimizde bir satış personelimizin, devre mülk sattığını ve satış tutarı üzerinden %1 prim aldığını farz edelim. Yapılan satış sonucunda İstanbul'daki suncuda yer alan veritabanına ait tablolarda peşisıra işlemler yapıldığını varsayalım.

Personelin bilgilerinin olduğu tabloda alacağı toplam prim tutarı satış tutarının %1'i oranında artsın , ödencek prime ait bilgiler ayrı bir tabloya işlensin ve aynı zamanda, şirkete ait finans kurumundaki toplam para hesabından bu prim tutarı kadar TL eksilsin. İşte bu üç işlemi göz önünde bulundurduğumuzda, tek bir işleme ait iş parçacıkları olduğunu anlayabiliriz. Dolayısıyla burada bir Transaction'dan rahatlıkla söz edebiliriz.Dilerseniz uygulamamıza geçelim. Öncelikle tablolarımıza bir göz atalım. IstanbulMerkez isimli veritabanımızda şu tablolar yer alıyor.



Şekil 2. IstanbulMerkez veritabanındaki tablolar.

Buradaki tablolardan ve görevlerinden kısaca bahsedelim. Personel tablosunda personele ait bilgiler yer alıyor. Bu tablo Prim isimli tablo ile bire-çok ilişkiye sahip. AFinans ise, bize ait finans kurumunun kasasındaki güncel TL'si miktarını tutan bir tablo. Personel satış yaptığında, satış tutarı üzerinde prim Personel tablosundaki PrimToplami alanının değerini %1 arttırıyor sonra Prim tablosuna bunu işliyor ve son olarakta AFinans tablosundaki Tutar alanından bahsi geçen %1 lik prim miktarını azaltıyor.

Peki buradaki üç işlem için neden Transaction kullanıyoruz? Farz edelim ki, Personel tablosunda PrimToplami alanının değeri arttıktan sonra, bir sorun nedeni ile veritabanına olan bağlantı kesilmiş olsun ve diğer işlemler gerçekleşmemiş olsun. Bu durumda personelin artan prim tutarını karşılayacak kadar TL'si finans kurumunun ilgili hesabından düşülmemiş olacağı için , finansal dengeler bozulmuş olacaktır. İşin içine para girdiği zaman Transaction'lar daha bir önem kazanmaktadır. Uygulamamızı basit olması açısından Console uygulaması olarak geliştireceğim. Haydi başlayalım. İşlemlerin kolay olması açısından başlangıç için Personel

tablosuna bir kayıt girdim. Şu an için görüldüğü gibi PrimToplami alanının değeri 0 TL'sidir. Ayıraca AFinans tablosunda bir başlangıç tutarımızın olması gerekiyor.

PersonelID	Ad	Soyad	Meslek	DogumTarihi	DogumYeri	PrimToplami
1	Hasmet	Samet	Matematik Mühendi	04.10.1980	Istanbul	0

Şekil 3. Personel Tablosu

ID	Tutar
1	10000000000000

Şekil 4. AFinans tablosu

Uygulamamızda , Personel tablosunda yer alan PrimToplami alanının değerini prim tutarı kadar arttırmak için aşağıdaki Stored Procedure'ü kullanacağız.

```
CREATE PROCEDURE [Prim Toplami Arttir]
@prim float,
@pid int
AS
UPDATE Personel SET PrimToplami = PrimToplami+@prim
WHERE PersonelID=@pid
GO
```

Prim tablosuna eklenecek veriler için ise INSERT sql cümlecigi içeren bir Stored Procedure'ümüz var.

```
CREATE PROCEDURE [Prim Bilgisi Gir]
@pid int,
@st float,
@p float,
@str datetime
AS
INSERT INTO Prim (PersonelID,SatisTutari,Prim,SatisTarihi)
VALUES (@pid,@st,@p,@str)
GO
```

Son olarak AFinans isimli tablomuzdan prim miktarı kadar TL'sini düşecek olan Stored Procedure'ümüzü yazalım.

```
CREATE PROCEDURE [Prim Dus]
@prim float
AS
UPDATE AFinans SET Tutar=Tutar-@prim
```

GO

Artık program kodlarımıza geçebiliriz. Kodları C# ile yazmayı tercih ettim.

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace TransactionSample1
{
    class Trans
    {
        [STAThread]
        static void Main(string[] args)
        {
            /* IstanbulMerkez veritabanına bir bağlantı nesnesi referans ediyoruz. */
            SqlConnection conIstanbulMerkez=new SqlConnection("initial catalog=IstanbulMerkez;data source=localhost;integrated
security=sspi");
            /* Transaction nesnemizi tanımlıyor ve bu Transaction'ın conIstanbulMerkez isimli SqlConnection nesnesinin belirttiği
bağlantıya ait komutlar için çalıştırılacağını belirtiyoruz. */
            SqlTransaction tr;
            conIstanbulMerkez.Open();
            tr=conIstanbulMerkez.BeginTransaction();
            double satisTutari=150000000000;
            double primTutari=satisTutari*0.01;
            /* Şimdi, Personel tablosundaki PrimToplami alanın değerini primTutari değişkeninin değerin kadar arttıracak Stored
Procedure'ü çalıştıracak SqlCommand nesnesini tanımlıyor ve gerekli parametreleri ekleyerek bu parametrelere değerlerini
veriyoruz. Son olarak da SqlCommand'in Transaction özelliğine oluşturduğumuz tr isimli SqlTransaction nesnesini atıyoruz. Bu şu
anlama geliyor. "Artık bu SqlCommand tr isimli Transaction içinde çalışacak olan bir iş parçacığıdır." */
            SqlCommand cmdPrimToplamiArttir=new SqlCommand("Prim Toplami Arttir",conIstanbulMerkez);
            cmdPrimToplamiArttir.CommandType=CommandType.StoredProcedure;
            cmdPrimToplamiArttir.Parameters.Add("@prim",SqlDbType.Float);
            cmdPrimToplamiArttir.Parameters.Add("@pid",SqlDbType.Int);
            cmdPrimToplamiArttir.Parameters["@prim"].Value=primTutari;
            cmdPrimToplamiArttir.Parameters["@pid"].Value=1;
            cmdPrimToplamiArttir.Transaction=tr;
            /* Aşağıdaki satırlarda ise "Prim Bilgisi Gir" isimli Stored Procedure'ü çalıştıracak olan SqlCommand nesnesi oluşturulup
gerekli paramtere ayarlamaları yapılıyor ve yine Transaction nesnesi belirlenerek bu komut nesneside Transaction bloğu içerisine
bir iş parçacığı olarak bildiriliyor.*/
            SqlCommand cmdPrimBilgisiGir=new SqlCommand("Prim Bilgisi Gir",conIstanbulMerkez);
            cmdPrimBilgisiGir.CommandType=CommandType.StoredProcedure;
```

```

cmdPrimBilgisiGir.Parameters.Add("@pid",SqlDbType.Int);
cmdPrimBilgisiGir.Parameters.Add("@st",SqlDbType.Float);
cmdPrimBilgisiGir.Parameters.Add("@p",SqlDbType.Float);
cmdPrimBilgisiGir.Parameters.Add("@str",SqlDbType.DateTime);
cmdPrimBilgisiGir.Parameters["@pid"].Value=1;
cmdPrimBilgisiGir.Parameters["@st"].Value=satisTutari;
cmdPrimBilgisiGir.Parameters["@p"].Value=primTutari;
cmdPrimBilgisiGir.Parameters["@str"].Value=System.DateTime.Now;
cmdPrimBilgisiGir.Transaction=tr;

```

/* Son olarak AFinans isimli tablodaki Tutar alanından prim tutarı kadar TL'sını düşecek olan Stored Procedure için bir SqlCommand nesnesi tanımlanıyor, prim tutarını taşıyacak olan parametre eklenip değeri veriliyor. Tabiki en önemlisi, bu komut nesnesi içinde SqlTransaction nesnemiz belirleniyor.*/

```

SqlCommand cmdTutarDus=new SqlCommand("Prim Dus",conIstanbulMerkez);
cmdTutarDus.CommandType=CommandType.StoredProcedure;
cmdTutarDus.Parameters.Add("@prim",SqlDbType.Float);
cmdTutarDus.Parameters["@prim"].Value=primTutari;
cmdTutarDus.Transaction=tr;

```

/* Evet sıra geldi programın can alıcı kodlarına. Aşağıda bir Try-Catch-Finally bloğu var. Bu bloklarda dikkat edecek olursanız tüm SqlCommand nesnelerinin çalıştırılması try bloğunda yapılmaktadır. Eğer tüm bu komutlar sorunsuz bir şekilde çalışırsa bu durumda, tr.Commit() ile transaction onaylanır vee değişikliklerin veritabanı üzerindeki tablolara yazılması onaylanmış olur*/

```

try
{
    int etkilenen=cmdPrimToplamiArttir.ExecuteNonQuery();
    Console.WriteLine("Personel tablosunda {0} kayıt güncellendi",etkilenen);
    int eklenen=cmdPrimBilgisiGir.ExecuteNonQuery();
    Console.WriteLine("Prim tablosunda {0} kayıt eklendi",eklenen);
    int hesaptaKalan= cmdTutarDus.ExecuteNonQuery();
    Console.WriteLine("AFinans tablosunda {0} kayıt güncellendi",hesaptaKalan);
    tr.Commit();
}

```

catch(Exception hata) /* Ancak bir hata olması durumdan ise, kullanıcı hatanın bilgisi ile uyarılır ve tr.Rollback() ile hatanın oluştuğu ana kadar olan tüm işlemler iptal edilir.*/

```

{
    Console.WriteLine(hata.Message+" Nedeni ile işlemler iptal edildi");
    tr.Rollback();
}

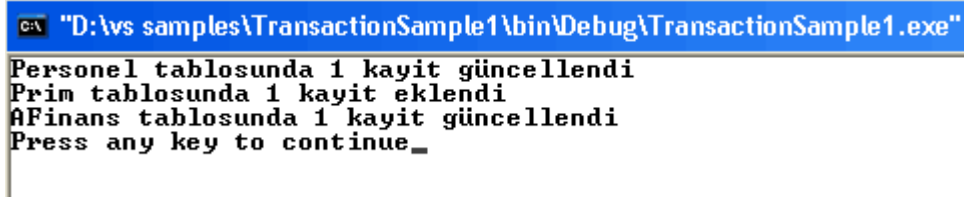
```

finally /* hata oluşmada oluşmasada açık bir bağlantı var ise bunun kapatılmasını garanti altına almak için finally bloğunda bağlantı nesnemizi Close metodu ile kapatırız.*/

```

    {
        conIstanbulMerkez.Close();
    }
}
}
}
}

```



Şekil 5. Programın çalışması sonucu.

Bu durumda veritabanındaki tablolara bakacak olursak; Personel tablosuna PrimToplami alanının değeri artmış,

PersonelID	Ad	Soyad	Meslek	DogumTarihi	DogumYeri	PrimToplami
1	Hasmet	Samet	Matematik Mühendisi	04.10.1980	Istanbul	1500000000

Şekil 6. Personel tablosunda PrimToplami alanının değeri artırıldı.

Prim tablosuna ilgili personele ait bilgiler eklenmiş,

PrimID	PersonelID	SatisTutari	Prim	SatisTarihi
6	1	150000000000	1500000000	17.11.2003 21:49:15

Şekil 7. Prim tablosuna ödenecek prime ait bilgiler girildi.

Son olarakta, AFinans isimli tablodaki Tutar alanının değeri güncellenmiştir.

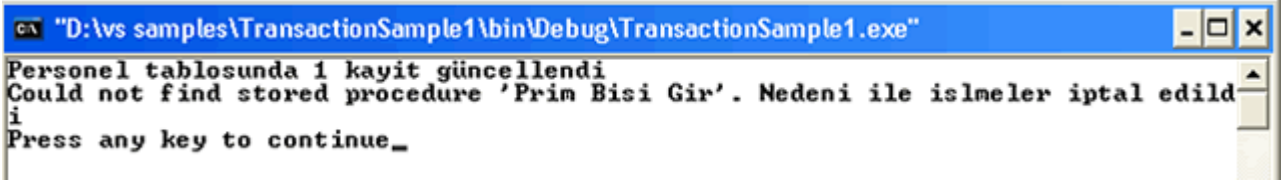
ID	Tutar
1	2498500000000

Şekil 8. AFinans tablosundaki Tutar alanının değeri prim tutarı kadar azaldı.

Şimdi dilerseniz bir hata tetikleyelim ve ne olacağına bakalım. Bir hata üretmek aslında isteyince o kadar kolay değil malesef. Ben Stored Procedure'lerden birinin ismini yanlış yazıcam. Bakalım ne olacak. Şu anda tablodaki veriler yukarıdaki Şekil 6,7 ve Şekil 8'de olduğu gibi.

```
SqlCommand cmdPrimBilgisiGir=new SqlCommand("Prim Bisi Gir",conIstanbulMerkez);
```

Burada Stored Procedure'ün ismi "Prim Bilgisi Gir" iken ben "Prim Bisi Gir" yaptım. Şimdi çalıştıracak olursak; aşağıdaki ekran ile karşılaşırız. İşlemler iptal edilir.



10. İşlemler iptal edildi.

Eğer transaction kullanmasaydık ilk SqlCommand çalışır ve Personel tablosunda PrimToplami alanın değeri artardı. Oysa şimdi bu tabloyu kontrol edicek olursak,

PersonelID	Ad	Soyad	Meslek	DogumTarihi	DogumYeri	PrimToplami
1	Hasmet	Samet	Matematik Mühendi	04.10.1980	Istanbul	1500000000

Şekil 11. Bir etki olmadı.

Tekrar önemli bir noktayı vurgulamak isterim. Bu Transaction tek bir veritabanı üzerinde çalışmaktadır. Eğer birden fazla veritabanı söz konusu olursa, Distributed Transaction tekniklerini kullanmamız gerekiyor. Bunu ilerleyen makalelerimizde incelemeye çalışacağım. Umarım buraya kadar anlattıklarım ile Transaction'lar hakkında bilgi sahibi olmuşsunuzdur. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Distributed Transactions

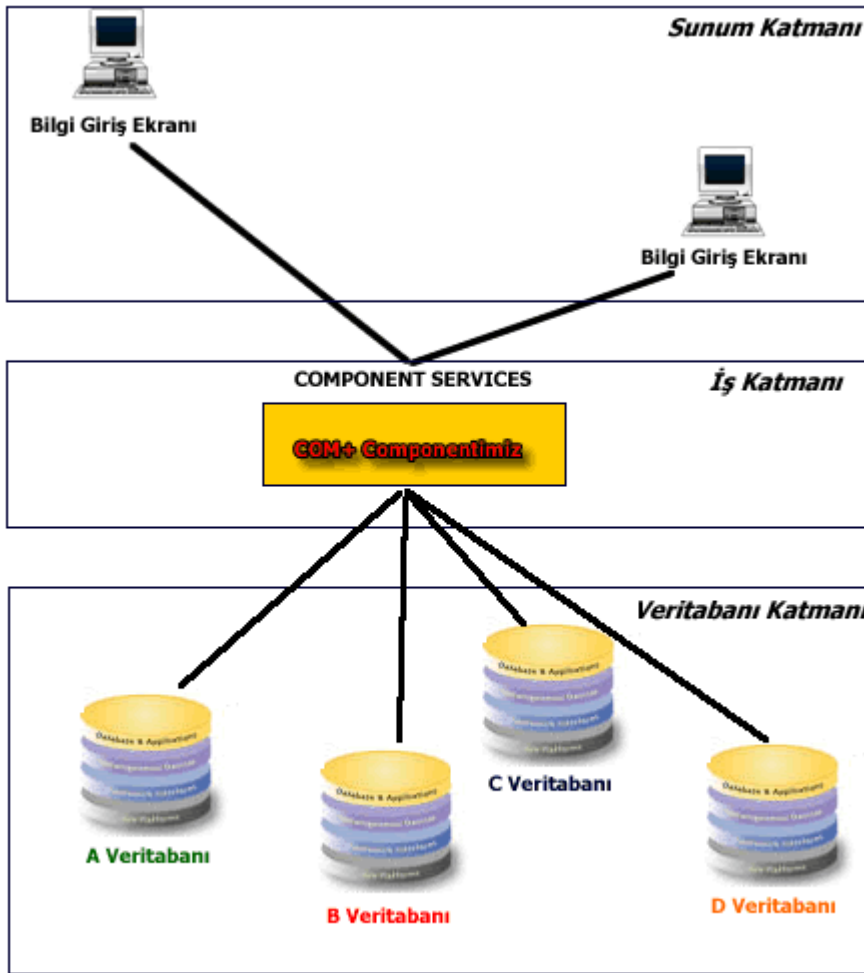
Değerli Okurlarım, Merhabalar.

Bildiğiniz gibi bir önceki makalemizde Transaction kavramından bahsetmiş, ancak birden fazla veritabanı için geçerli olacak Transaction işlemlerinin Dağıtık Transaction'lar olarak adlandırıldığından söz etmiştik. Bu makalemizde Dağıtık Transaction'ları inceleyecek ve her zaman olduğu gibi konuyu açıklayıcı basit bir örnek geliştireceğiz.

İş uygulamalarında, **Online Transaction Processing(OLTP)** dediğimiz olay çok sık kullanılmaktadır. Buna verilecek en güzel örnek bankaların ATM uygulamalarıdır. Veriler eş zamanlı olarak aynı anda bir bütün halinde işlenmekte ve güncellenmektedir. Bu tarz projelerin uygulanmasında OLTP tekniği yaygın bir biçimde kullanılmaktadır. Bu tekniğin uygulanabilmesi Dağıtık Transaction'ların kullanılmasını gerektirir. .NET ile Dağıtık Transaction'lar yazmak için **Component Services**'i kullanmamız gerekmektedir. Özellikle, çok katlı mimari dediğimiz iş uygulamalarında Dağıtık Transaction'ları çok sık kullanırız. Burada Dağıtık Transaction'lar başka componentler tarafından üstlenilir ve Sunu Katmanı ile Veritabanları arasındaki işlemlerin gerçekleştirilmesinde rol oynarlar. Bu component'lerin bulunduğu katman İş Katmanı olarak da adlandırılmaktadır. Nitekim Componentler aslında Transaction başlatıp gerekli metodları çalıştırarak veriler üzerindeki bütünsel işlevleri yerine getirir ve transaction'ı sonlandırır. Yani Sunum Katmanı'nda yer alan uygulamalar sadece gerekli verileri parametre olarak iş katmanında yer alan componentlere gönderirler. Dolayısıyla üzerlerinde ilgili veritabanı verileri için herhangi bir fonksiyon veya metodun çalıştırılmasına gerek yoktur. Bütün sorumluluk Component Services 'da yer alan COM+ componentindedir. Burada veritabanlarına

bağlanılır , gerekli düzenlemeler bir Transaction içerisinde gerçekleştirilir ve sorunsuz bir transaction tüm iş parçacıkları ile teyid edilerek gerekli düzenlemeler veritabanlarına yansıtılır.

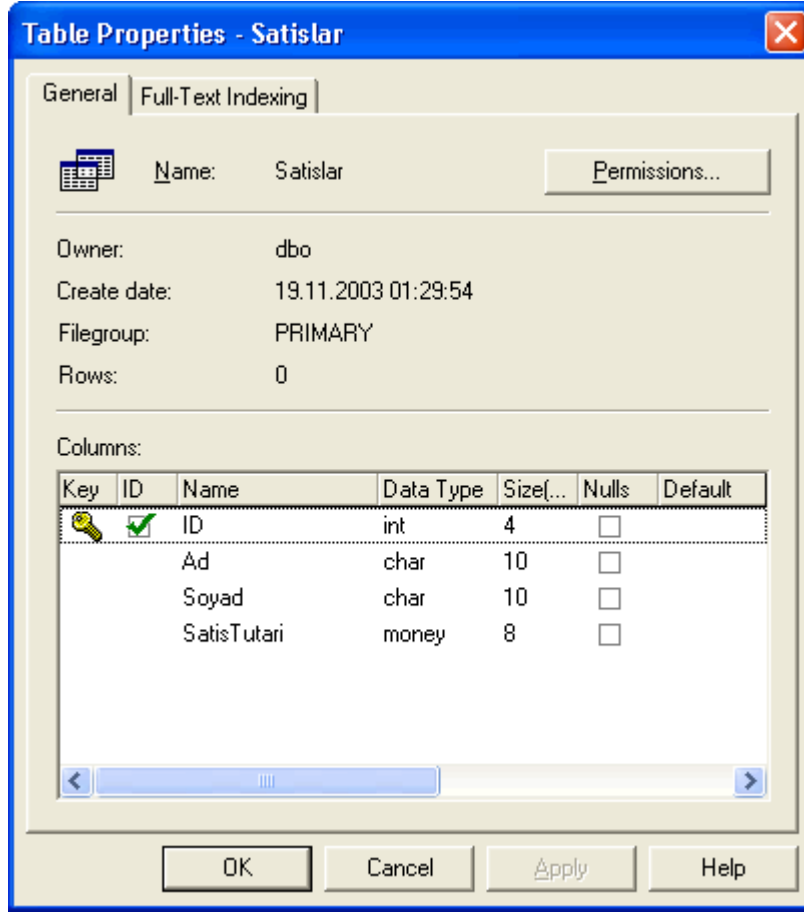
En basit haliyle 3 katlı mimaride, Sunum Katmanı ile Veritabanları arasındaki transaction işlemlerini COM+ Component'leri ile gerçekleştirebiliriz. Bu component'ler Windows 2000' den itibaren Component Service olarak adlandırılan bir servis altında yer almaktadırlar. Elbetteki bu component'i biz geliştireceğiz. Component'in görevi , transaction işlemlerinin otomatik yada manuel olarak gerçekleştirilmesini sağlamaktır. Bir dll dosyası haline getirilen bu component'leri istenilen Sunum Katmanı uygulamasına ekleyerek kullanabiliriz. Yazacağımız component içinde Transaction işlemlerini kullanabilmek amacıyla .NET içerisinde yer alan **System.EnterpriseServices** sınıfının metodlarını kullanırız. Oluşturulan component'i örneklerimizde de göreceğiniz gibi bir **Strong Name** haline getirmemiz gerekmektedir. Örneğimizi yazarken bunları daha iyi anlıyacaksınız.Üç katlı mimaride, Dağıtık Transaction uygulamalarının aşağıdaki şekil ile zihninizde daha berrak canlanacağı kanısındayım.



Şekil 1. 3 Katlı Mimari için COM+ Kullanımı

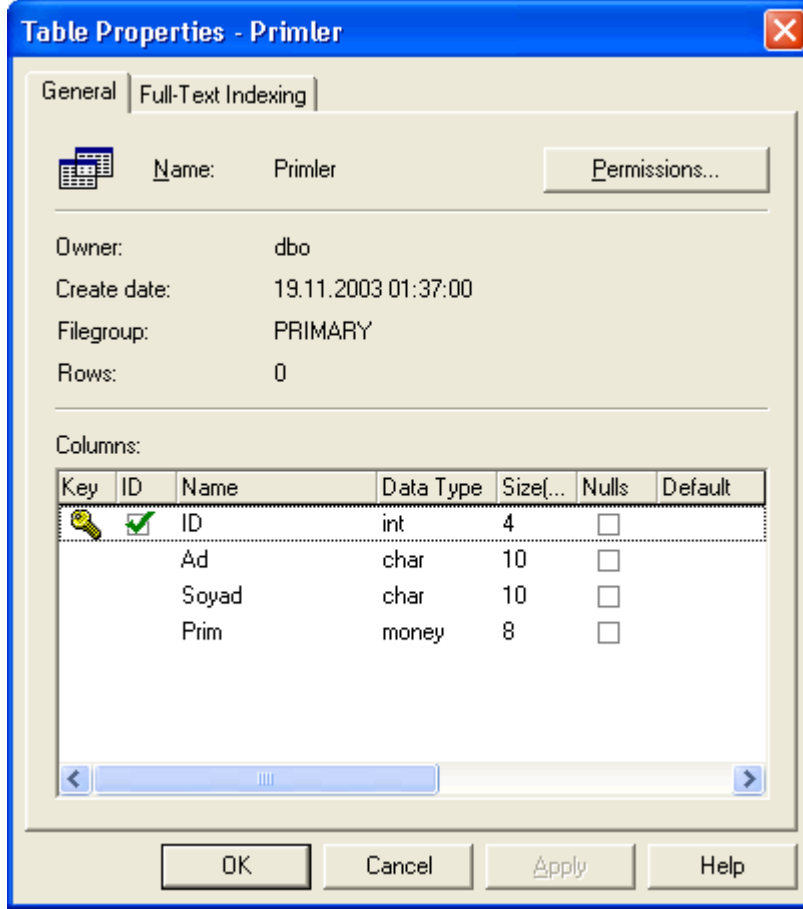
Özetlemek gerekirse, Dağıtık Transaction' ların kullanıldığı uygulamalarda Component Services kullanılması gerekmektedir. Bir Dağıtık Transaction Component'i yazdığımızda, transaction işlemlerini otomatik olarak Component Service'a yaptırabiliriz. Bunun yanında **ContextUtil** adı verilen nesneyi ve bu nesneye ait **SetComplete()**, **SetAbort()** gibi metodları kullanarak Transaction işlemlerini elle de yapılandırabiliriz. Bu makalemizde otomatik olanı seçtim. Bir sonraki makalede işlemleri manuel olarak

yapacağız.Dilerseniz örneğimize geçelim . Bu uygulamızda 2 veritabanı üzerindeki 2 farklı tablo için, bir Transaction içerisinde basit veri giriş işlemleri gerçekleştireceğiz. Öncelikle tablolarımızın yapısını ve databaselerimizi belirtelim. Firends isimli bir Veritabanı' nda kullanacağımız basit bir tablo var. Satıslar isimli bu tabloda Ad,Soyad ve SatisTutari alanları yer almakta.



Şekil 2. Satıslar tablosunun yapısı.

İkinci Veritabanımız ise İstanbulMerkez. Tablolarımızın adı Primler. Bu tabloda ise yine Ad,Soyad ve Prim bilgisi yer alıyor.



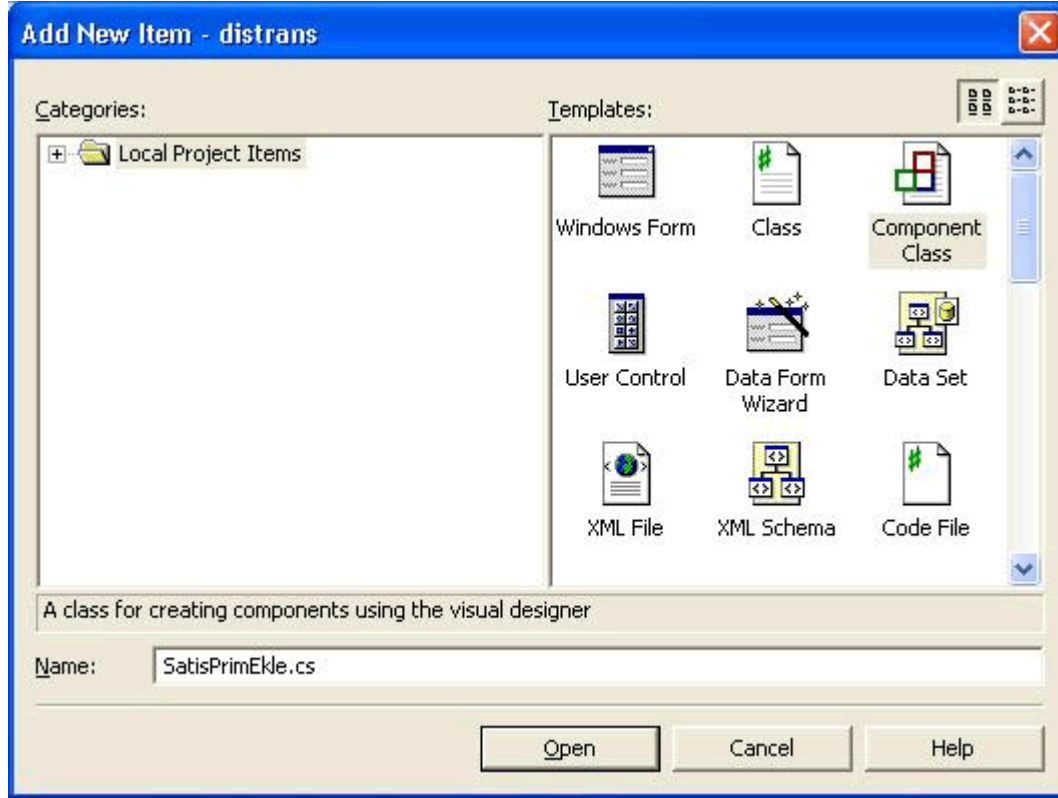
Şekil 3. Primler tablosunun yapısı.

Uygulamamızda Satıslar isimli tabloya bilgi girildikten sonra SatisTutari'nin %10' u üzerinden prim hesaplanacak ve aynı anda Primler tablosuna bu bilgiler eklenecek. Tabiki bu iki basit veritabanı işlemi bir Transaction içinde gerçekleştirilecek. Uygulamamızı tasarlamaya başlayalım. Önce yeni bir C# ile yeni bir **Windows Application** oluşturalım. Bu uygulamanın içerdiği Form Sunum Katmanı'nda yer alan veri giriş ekranımız olacaktır. Formu aşağıdakine benzer veya aynı şekilde tasarlayalım.



Şekil 4. Formun yapısı.

Kullanıcı bu ekrandan Ad,Soyad ve Satış Tutarı bilgilerini girecek. Girilen bu bilgiler, yazacağımız COM+ Componentindeki metoda parametre olarak gidicek ve bu metod içinde işlenerek veritabanlarındaki tablolarda gerekli düzenlemeler yapılacaktır. Eğer tüm işlemler başarılı olursa ve metod tam olarak çalışırsa geriyede işlemlerin dolayısıyla Transaction'ın başarılı olduğuna dair bir string bilgi gönderecek. Evet şimdi uygulamanın en önemli kısmına sıra geldi . Componentin tasarlanmasına. İlk önce, Project menüsünden **Add Component** komutunu vererek component' imizi uygulamamıza ekliyoruz.



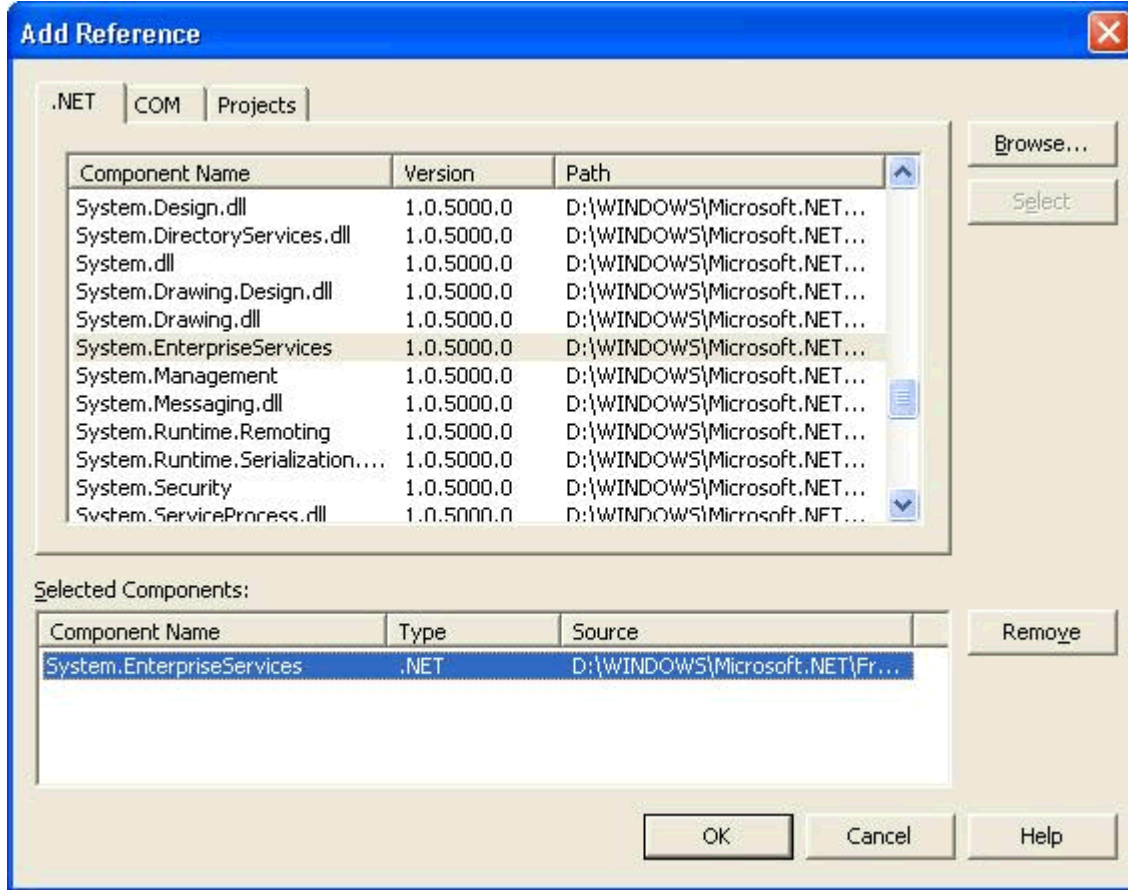
Şekil 5. Component Ekleme.

Ben componentimize SatisPrimEkle adını verdim. Bu durumda Solution'ımıza SatisPrimEkle.cs isimli dosya eklenir ve Visual Studio.NET IDE'de aşağıdaki gibi görünür.



Şekil 6. Componentin ilk eklendiğinde IDE’de durum.

Şimdi ise bu component içersinde yer alacak dağıtık transaction işlemleri için gerekli olan referansımızın projemize eklememize gerekiyor. Daha öncede System.EnterpriseServices olarak bahsettiğimiz bu sınıfı eklemek için yine, Project menüsünden, **Add Reference** komutunu veriyoruz. Burada ise .NET sekmesinden **System.EnterpriseServices** sınıfını ekliyoruz.



Şekil 7. System.EnterpriseServices Sınıfının eklenmesi.

Şimdi Componentimizin kodlarını yazmaya başlayabiliriz. **To Switch To Code Window** linkine tıklayarak component'imizin kodlarına geçiş yapıyoruz. İlk haliye kodlar aşağıdaki gibidir.

```
using System;
using System.ComponentModel;
using System.Collections;
using System.Diagnostics;
namespace distrans
{
    /// <summary>
    /// Summary description for SatisPrimEkle.
    /// </summary>
```

```
public class SatisPrimEkle : System.ComponentModel.Component
{
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.Container components = null;
    public SatisPrimEkle(System.ComponentModel.IContainer container)
    {
        ///
        /// Required for Windows.Forms Class Composition Designer support
        ///
        container.Add(this);
        InitializeComponent();
        //
        // TODO: Add any constructor code after InitializeComponent call
        //
    }
    public SatisPrimEkle()
    {
        ///
        /// Required for Windows.Forms Class Composition Designer support
        ///
        InitializeComponent();
        //
        // TODO: Add any constructor code after InitializeComponent call
        //
    }
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    protected override void Dispose( bool disposing )
    {
        if( disposing )
        {
            if(components != null)
            {
                components.Dispose();
            }
        }
    }
}
```

```

        base.Dispose( disposing );
    }
    #region Component Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        components = new System.ComponentModel.Container();
    }
    #endregion
}
}

```

Biz buradaki kodları aşağıdaki şekliyle düzenleyecek ve yukarıda yazılı çoğu kodu çıkartacağız. Haydi başlayalım. Öncelikle using kısmına,

```

using System.Data.SqlClient;

using System.EnterpriseServices;

```

sınıflarını eklememiz gerekiyor. Çünkü Transaction işlemleri için **EnterpriseServices** sınıfını ve veritabanı işlemlerimiz içinde **SqlClient** sınıfında yer alan nesnelerimizi kullanacağız. İkinci köklü değişim ise SatisPrimEkle isimli sınıfımızın **ServicedComponent** sınıfından türetilmiş olması. Bu değişikliği ve diğer fazlalıkları çıkarttığımız takdirde, kodlarımızın son hali aşağıdaki gibi olacaktır.

```

using System;
using System.ComponentModel;
using System.Collections;
using System.Diagnostics;
using System.Data.SqlClient;
using System.EnterpriseServices;
namespace distrans
{
    public class SatisPrimEkle : ServicedComponent
    {
    }
}

```

Şimdi metodumuzu ekliyelim ve gerekli kodlamaları yazalım.

```

using System;
using System.ComponentModel;
using System.Collections;
using System.Diagnostics;
using System.Data.SqlClient;
using System.EnterpriseServices;
namespace distans
{

```

/* [Transaction(TransactionOption.Required)] satırı ile belirtilen şudur. Component' imiz var olan Transaction içerisinde çalıştırılacaktır. Ancak eğer oluşturulmuş yani başlatılmış bir transaction yoksa, bu component' imiz için yeni bir tane oluşturulması sağlanacaktır. Burada,

TransactionOption'ın sahip olabileceği diğer değerler Disabled, NotSupported, RequiresNew ve Supported dir.

Disabled durumunda, transaction özelliği görmezden gelinir. Default olarak bu değer kabul edilir. Bu durumda Transaction başlatılması gibi işlemler manuel olarak yapılır.

Not Supported durumunda ise Component' imiz bir transaction var olsa bile bu transaction'ın dışında çalışacaktır.

RequiresNew durumunda, Component' imiz için bir transaction var olsada olmasada mutlaka yeni bir transaction başlatılacaktır.

Supported durumu ise , var olan bir transaction olması durumunda, Component' imizin bu transaction'a katılmasını sağlar.

Biz uygulamamızda otomatik transaction tekniğini kullandığımız için Required seçeneğini kullanıyoruz.

*/

```

[Transaction(TransactionOption.Required)]public class SatisPrimEkle : ServicedComponent
{

```

/* AutoComplete() satırı izleyen metodun bir transaction içerisinde yer alacağını ve transaction işlemlerinin başlatılması ve bitirilmesini Component Services 'ın üstleneceğini belirtir. Dolayısıyla Component' imizin bu metodunu çalıştırdığımızda bir transaction başlatılır ve ContextUtil nesnesi ile manuel olarak yapacağımız SetComplete (Commit) ve SetAbort(Rollback) hareketlerini COM+ Servisi kendisi yapar. */

```

[AutoComplete()]public string VeriGonder(string ad,string soyad,double satisTutari)
{

```

```

    SqlConnection conFriends = new SqlConnection("initial catalog=Friends;data source=127.0.0.1;integrated
security=sspi;packet size=4096");

```

```

    SqlConnection conIstanbulMerkez = new SqlConnection("initial catalog=IstanbulMerkez;data
source=127.0.0.1;integrated security=sspi;packet size=4096");

```

/* Yukarıdaki SqlConnection nesneleri tanımlanırken data source özelliklerine sql sunucusunun bulunduğu ip adresi girildi. Bu farklı ip'lere sahip sunucular söz konusu olduğunda farklı veritabanlarının da kullanabiliriz anlamına gelmektedir. Uygulamamızı aynı sunucu üzerinde gerçekleştirmek zorunda olduğum için aynı ip adreslerini verdim.*/

/* Aşağıdaki satırlarda veri girişi için gerekli sql cümlelerini hazırladık ve bunları SqlCommand nesneleri ile ilişkilendirip çalıştırdık. */

```

    string sql1="INSERT INTO Satislar (Ad,Soyad,SatisTutari) VALUES ('"+ad+"','"+soyad+"','"+satisTutari+"')";
    double prim=satisTutari*0.10;
    string sql2="INSERT INTO Primler (Ad,Soyad,Prim) VALUES ('"+ad+"','"+soyad+"','"+prim+"')";

```

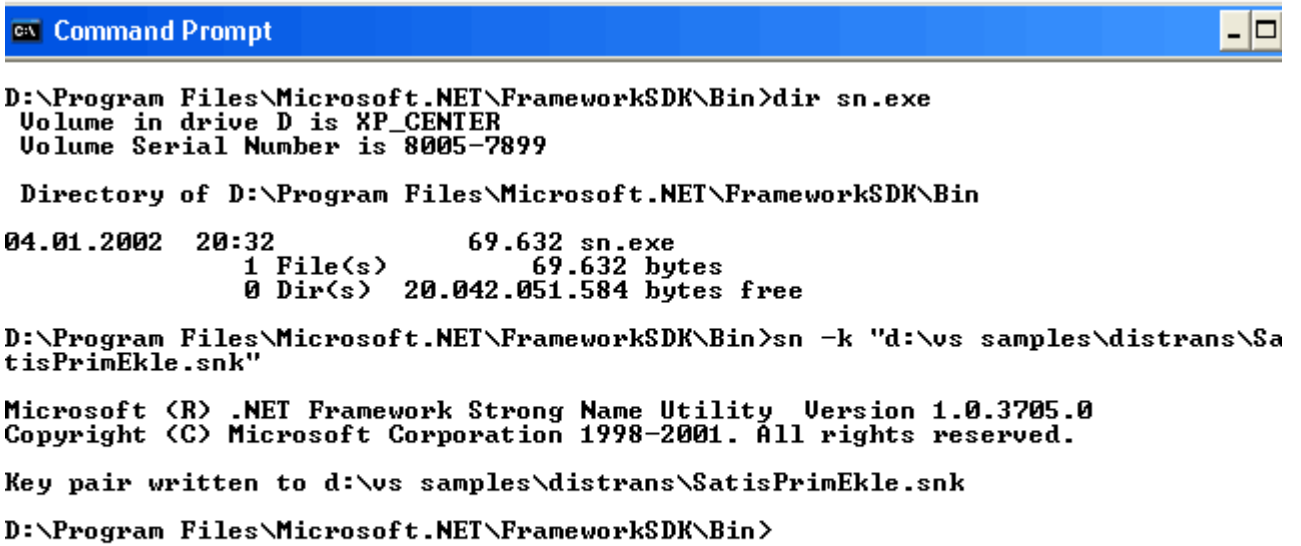


```

SqlCommand cmdSatisGir=new SqlCommand(sql1,conFriends);
SqlCommand cmdPrimGir=new SqlCommand(sql2,conIstanbulMerkez);
conFriends.Open();
conIstanbulMerkez.Open();
cmdSatisGir.ExecuteNonQuery();
cmdPrimGir.ExecuteNonQuery();
return "ISLEM TAMAM"; /* Metod başarılı bir şekilde çalıştığında, COM+ seviyesi transaction'ı otomatik olarak sonlandırır ve
metodumuz geriye ISLEM TAMAM stringini döndürür. */
    }
}
}

```

Component' imizi oluşturduktan sonra bunu Sunum Katmanındaki uygulamalarda kullanabilmek ve COM+ Servisi'ne eklenmesini sağlamak için bir **Strong Name Key** dosyası oluşturmamız gerekiyor. Bu IDE dışından ve **sn.exe** isimli dosya ile yapılan bir işlemdir. Bunun için D:\Program Files\Microsoft.NET\FrameworkSDK\Bin\sn.exe dosyasını kullanacağız. İşte komutun çalıştırılışı;



```

C:\> Command Prompt

D:\Program Files\Microsoft.NET\FrameworkSDK\Bin>dir sn.exe
Volume in drive D is XP_CENTER
Volume Serial Number is 8005-7899

Directory of D:\Program Files\Microsoft.NET\FrameworkSDK\Bin

04.01.2002  20:32                69.632 sn.exe
             1 File(s)                69.632 bytes
             0 Dir(s)  20.042.051.584 bytes free

D:\Program Files\Microsoft.NET\FrameworkSDK\Bin>sn -k "d:\vs samples\distrans\SatisPrimEkle.snk"

Microsoft (R) .NET Framework Strong Name Utility  Version 1.0.3705.0
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.

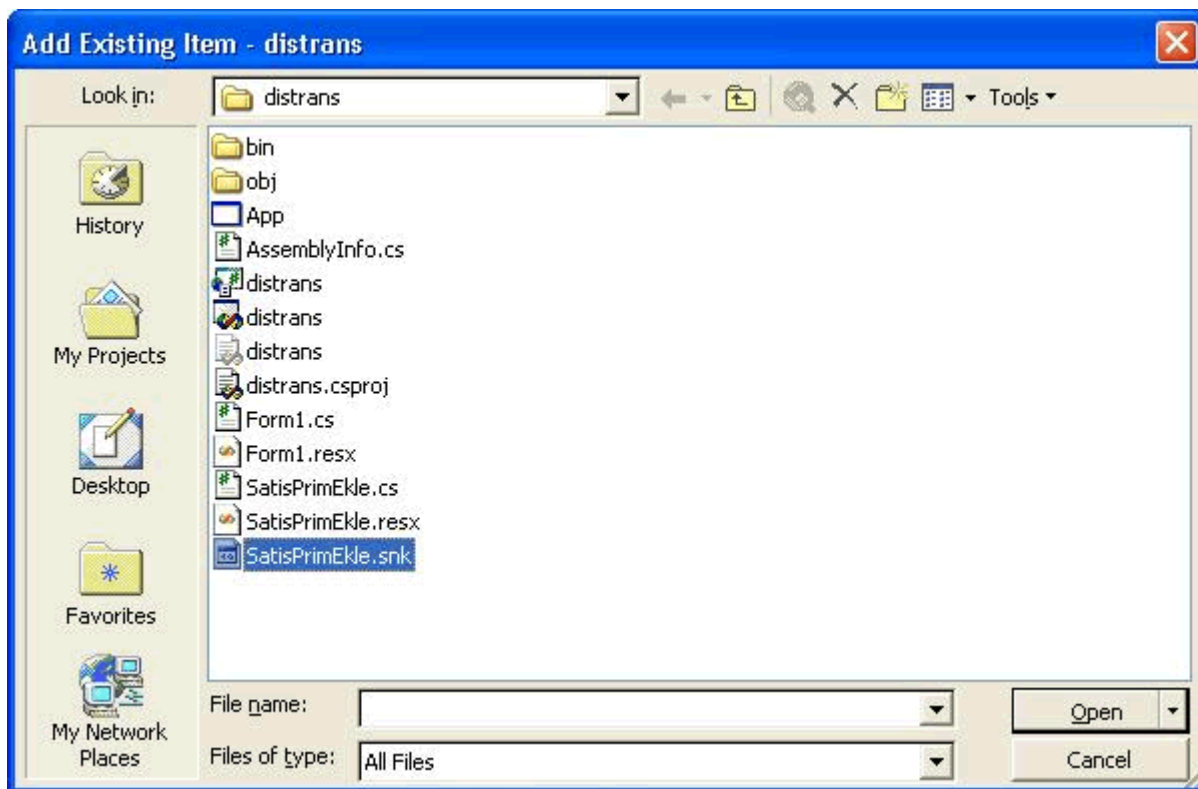
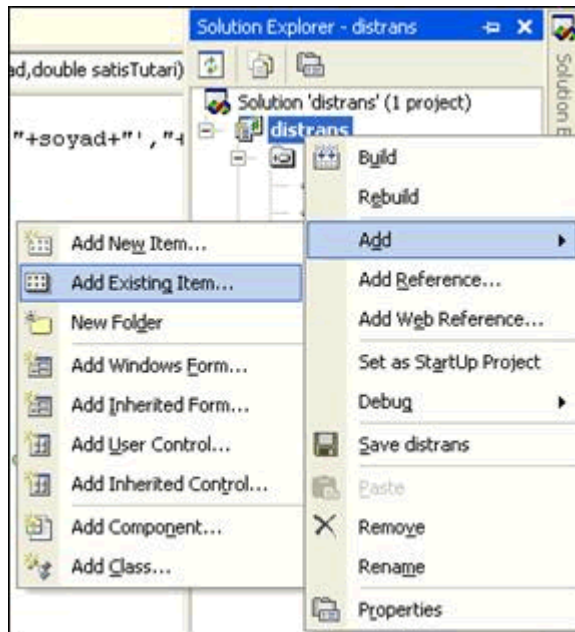
Key pair written to d:\vs samples\distrans\SatisPrimEkle.snk

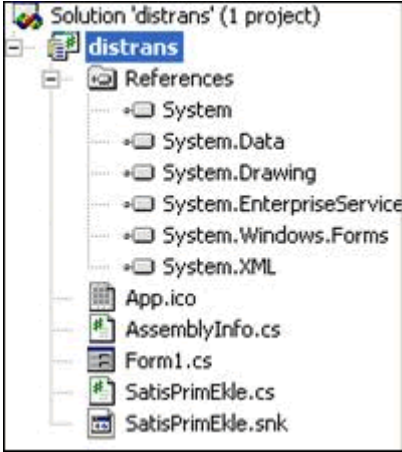
D:\Program Files\Microsoft.NET\FrameworkSDK\Bin>

```

Şekil 8. sn.exe ile snk(strong name key) dosyasının oluşturulması.

Görüldüğü gibi **snk uzantılı** dosyamız oluşturuldu. Şimdi Formumuzda bu Component'imize ait metodu kullanabilmek için oluşturulan bu snk uzantılı dosyayı Solution'ımıza **Add Exciting Item** seçeneği ile eklememiz gerekiyor. Aşağıdaki 3 şekilde bu adımları görebilirsiniz.





Şekil 9 . SatisPrimEkle.snk dosyasının projemize eklenmesi.

Formumuzda Component'imize ait metodu kullanabilmek için yapmamız gereken bir adım daha var. Oda uygulamanın **AssemblyInfo.cs** dosyasına aşağıdaki kod satırını eklemek.

```
[assembly: AssemblyKeyFile("..\\..\\SatisPrimEkle.snk")]
```

Şimdi formumuzdaki kodları inceleyelim. Öncelikle SatisPrimEkle tipinde bir nesne tanımlıyoruz. Şimdi bu nesnesin VeriGonder isimli metoduna eriştiğimizde aşağıdaki şekilde gördüğümüz gibi IntelliSense özelliği bize kullanabileceğimiz parametreleride gösteriyor.

```
private void btnGonder_Click(object sender, System.EventArgs e)
{
    SatisPrimEkle comp=new SatisPrimEkle();

    double st=System.Convert.ToDouble(txtSatisTutari.Text);

    comp.Verigonder (|
)... string SatisPrimEkle.Verigonder (string ad, string soyad, double satisTutari)
```

Şekil 10. Metodun kullanımı.

Şimdi tüm kodumuzu tamamlayalım ve örneğimizi çalıştıralım.

```
private void btnGonder_Click(object sender, System.EventArgs e)
{
    SatisPrimEkle comp=new SatisPrimEkle();
    double st=System.Convert.ToDouble(txtSatisTutari.Text);
    try
    {
        MessageBox.Show(comp.Verigonder(txtAd.Text,txtSoyad.Text,st));
    }
}
```

```

catch(Exception hata)
{
    MessageBox.Show(hata.Source + ":" + hata.Message);
}
}

```

Şimdi örneğimizi çalıştıralım.



Şekil 11. ISLEM TAMAM

Görüldüğü gibi , metodumuz başarılı bir şekilde çalıştırıldı. Hemen tablolarımızı kontrol edelim.

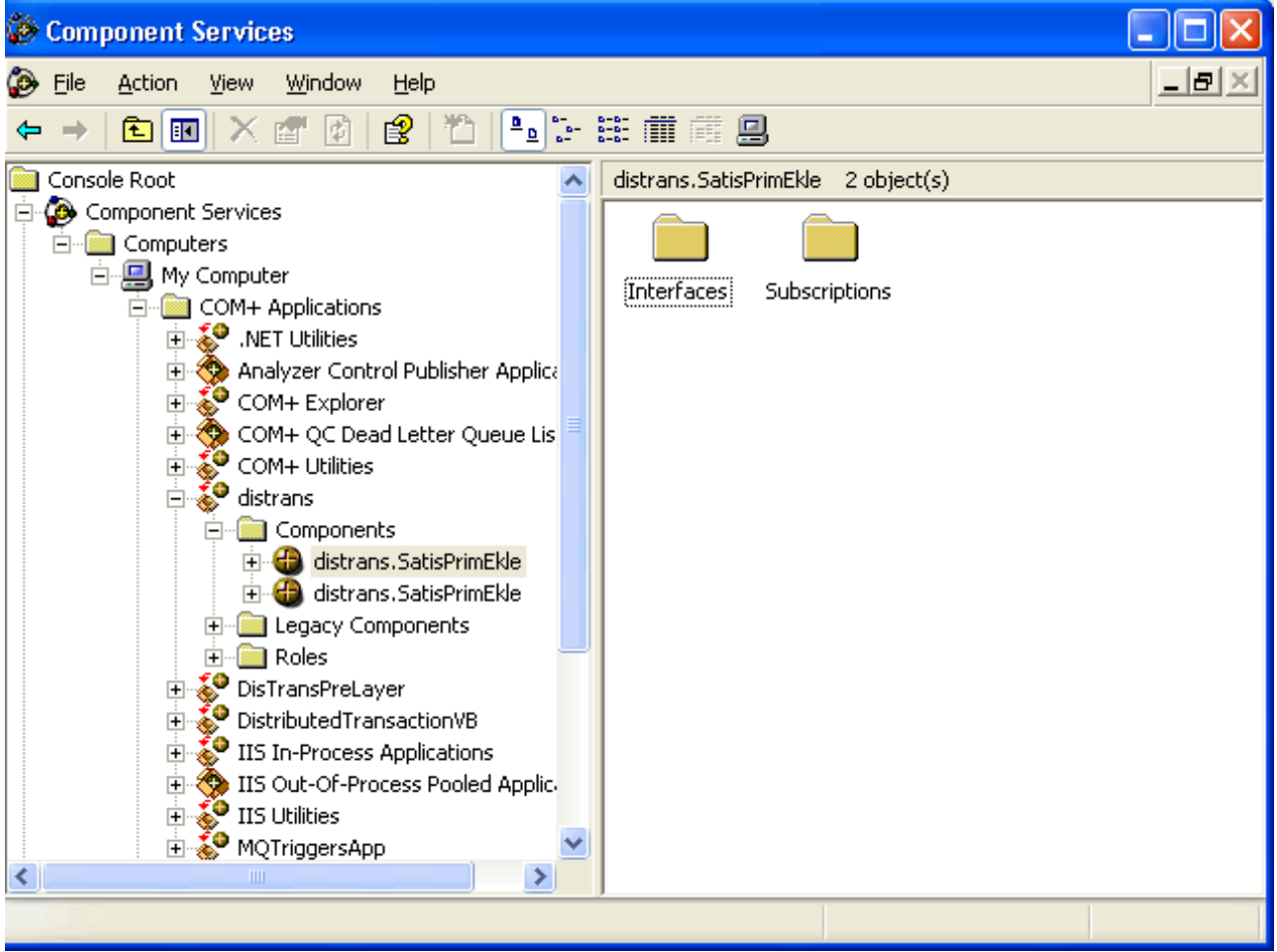
	ID	Ad	Soyad	Prim
1	1	Burak	Senyurt	150000000
*				

Şekil 12. IstanbulMerkez veritabanındaki Primler Tablosu.

Data in Table 'Satislar' in 'Friends' on '(local)'				
	ID	Ad	Soyad	SatisTutari
1	1	Burak	Senyurt	150000000
*				

Şekil 13. Friends veritabanındaki Satislar tablosu.

Şimdi **Component Services**'a bakacak olursak oluşturmuş olduğumuz distrans isimli Component Application'ı ve içerdiği SatisPrimEkle isimli Component'i, burada da görebiliriz.



Şekil 14. Componet Services.

Bir sonraki makalemizde aynı örneği Manuel yöntemlerle ve dolayısıyla ContextUtil sınıfının metodları ile gerçekleştireceğiz. Hepinize mutlu günler dilerim.

Bir Form ve Kontrollerinin Elle Programlanması

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, bir Formu kodla nasıl oluşturacağımızı, bu form üstüne nasıl kontroller ekleyeceğimizi, bu kontroller için nasıl olaylar yazacağımızı vb. konuları işlemeye çalışacağız. Bildiğiniz gibi Visual Studio.NET gibi grafiksel ortamlar ile Form ve Form nesnelerini görsel olarak, kolay ve hızlı bir şekilde oluşturabilmekteyiz. Bu bizim programlama için ayıracağımız sürede , ekran tasarımlarının daha hızlı yapılabilmesine olanak sağlamaktadır.

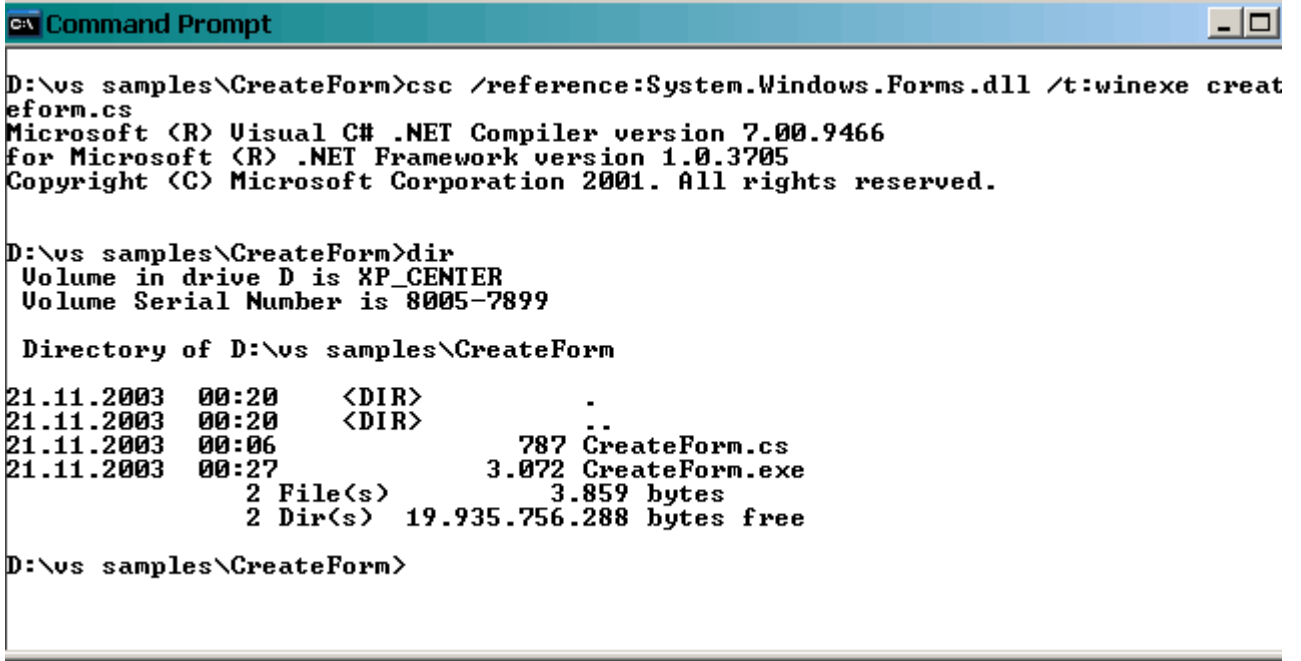
Ancak bazen elimizde sadece csc gibi bir C# derleyicisi ve .Net Framework vardır. İşte böyle bir durumda, Windows Form'larını tasarlamak için manuel olarak kodlama yapmamız gerekmektedir. Ayrıca, iyi ve uzman bir programcı olabilmek için özellikle Visual ortamlarda Windows Form, Button, TextBox gibi kontrollerin nasıl oluşturulduğunu, nasıl kodlandığını olaylara nasıl ve ne şekilde bağlandığını bilmek, oldukça faydalıdır. Bu aynı zamanda kontrolün bizde olmasını da sağlayan bir unsur olarak karşınıza çıkar ve kendimize olan güvenimizi dahada artırır.

Dilerseniz konuyu anlamak için basit ama etkili bir örnekle başlayalım. Bu örneğimizde basit olarak boş bir Form oluşturacağız ve bunu csc.exe (C# Compiler) ile derleyeceğiz. Bir Windows Formu aslında System.Windows.Forms sınıfından türeyen bir nesnedir. Bu nedenle oluşturacağımız C# sınıfı içerisinde bu bildirimi gerçekleştirmeliyiz. Ayrıca sınıfımızın Form nesnesine ait elemanlarda kullanabilmesi için, Form sınıfından türetmeliyiz(Inheriting). Bunlara ek olarak Formumuzu ekranda gösterebilmek için Application nesnesini ve buna bağlı Run metodunu kullanacağız. Hemen bir text editor açalım ve burada aşağıdaki kodları girelim.

```
using System.Windows.Forms;
public class BirForm:Form // Form sınıfının elemanlarını kalıtısal olarak devralıyoruz.
```

```
{
    public static void Main() // Programın başladığı nokta.
    {
        BirForm yeni1=new BirForm(); // BirForm sınıfından bir nesne tanımlıyoruz.
        Application.Run(yeni1);
        /* yeni1 isimli Form nesnemiz Application nesnesi tarafından görüntülenir. Bu noktadan itibaren programın işleyişi bir
        döngüye girer. Bu döngüde Application nesnesi sürekli olarak programın işleyişini sonlandıracak bir olayın tetiklenip tetiklenmediğini
        de kontrol eder. Bu arada tabi yazılan diğer olaylar ve metodlar çalışır. Ancak program sonlandırma ile ilgili ( örneğin Close metodu
        ) bir kod ile karşılaşıldığında veya kullanıcı Form'un varsa kapatma simgesine tıkladığında (veya ALT+F4 yaptığında) Application
        nesnesi artık programın işleyişini durdurur. */
    }
}
```

Yazdığımız bu dosyayı cs uzantısı ile kaydetmeyi unutmayalım. Şimdi bu dosyayı csc.exe ile derleyelim. Programı derlerken dikkat etmemiz gereken bir nokta var. System.Windows.Forms ` un programa referans edilmesi gerekir. Bunu sağlamak için derleme komutumuzun nasıl yazıldığına dikkat edelim. /reference: burada devreye girmektedir.



```

C:\ Command Prompt

D:\vs samples\CreateForm>csc /reference:System.Windows.Forms.dll /t:winexe creat
eform.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

D:\vs samples\CreateForm>dir
Volume in drive D is XP_CENTER
Volume Serial Number is 8005-7899

Directory of D:\vs samples\CreateForm

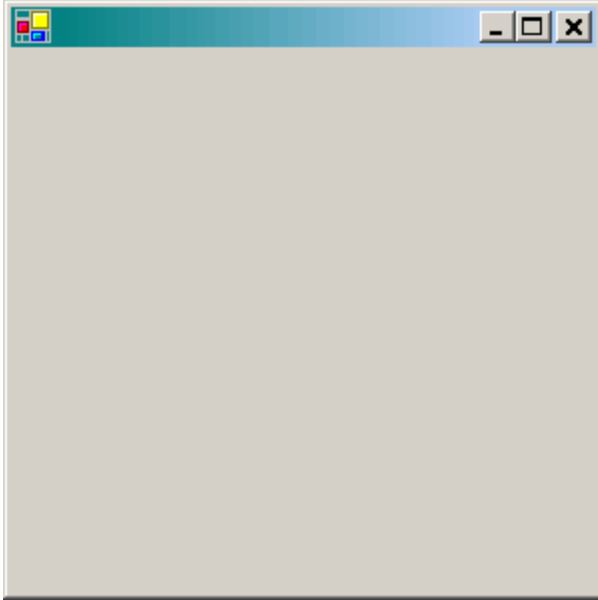
21.11.2003  00:20    <DIR>          .
21.11.2003  00:20    <DIR>          ..
21.11.2003  00:06                787 CreateForm.cs
21.11.2003  00:27            3.072 CreateForm.exe
                2 File(s)              3.859 bytes
                2 Dir(s) 19.935.756.288 bytes free

D:\vs samples\CreateForm>

```

Şekil 1. İlk Derleme

Görüldüğü gibi, csc dosyamızı derlemiş ve CreateForm.exe dosyasını oluşturmuştur. Burada /t:winexe programı Windows işletim sistemine, " Ben bir WinForm'um " olarak tanıtmaktadır. Şimdi bu dosyayı komut satırından çalıştıracak olursak aşağıdaki şekilde görülen sonucu elde ederiz.



Şekil 2. Oluşturulan Form Nesnemiz.

Şekil 2.'de oluşturulmuş olduğumuz Form nesnesini görebilirsiniz. Yazmış olduğumuz kodlarda bu Form nesnesine ait özellikleri değiştirerek farklı Form görünümüleride elde edebiliriz. Bu noktada size Form oluşturma olaylarının kodlama tekniğinin aslen nasıl yapılması gerektiğini göstermek isterim. Bu bir tarzdır yada uygulanan bir formattır ancak en uygun şekildir. Nitekim Visual

Studio.NET ortamında bir Windows Application geliştirdiğinizde, uygulama kodlarına bakacak olursanız bahsetmiş olduğumuz formasyonun uygulanmış olduğunu göreceksiniz.

```
using System.Windows.Forms;
public class BirForm:Form
{
    public BirForm() // Constructor(Yapıcı) metodumuz.
    {
        InitializeComponent(); /* BirForm sınıfından bir Form nesnesi üretildiğinde new yapılandırıcısı bu constructora bakar ve
        InitializeComponent metodunu çağırır. */
    }
    private void InitializeComponent()
    {
        /* Burada Form'a ait özellikler ve Form üzerinde yer alacak nesneler tanılanır. Tanılanan nesneler aynı zamanda Form'a
        burada eklenir ve özellikleri belirlenir. */
    }
    public static void Main()
    {
        Application.Run(new BirForm());
    }
}
```

Buyazım tekniği daha anlamlı değil mi ? Kodumuzu bu şekilde değiştiripçalıştırdığımızda yine aynı sonucu elde ederiz. Ancak dilersek bu kodu şu şekilde de yazabiliriz.

```
using System.Windows.Forms;
public class BirForm:Form
{
    public BirForm()
    {
        NesneleriAyarla();
    }
    private void NesneleriAyarla()
    {
    }
    public static void Main()
    {
        Application.Run(new BirForm());
    }
}
```


Yine aynı sonucu alırız.Şimdi Formumuza biraz renk katalım ve üstünede bir kaç nesne ekleyelim.

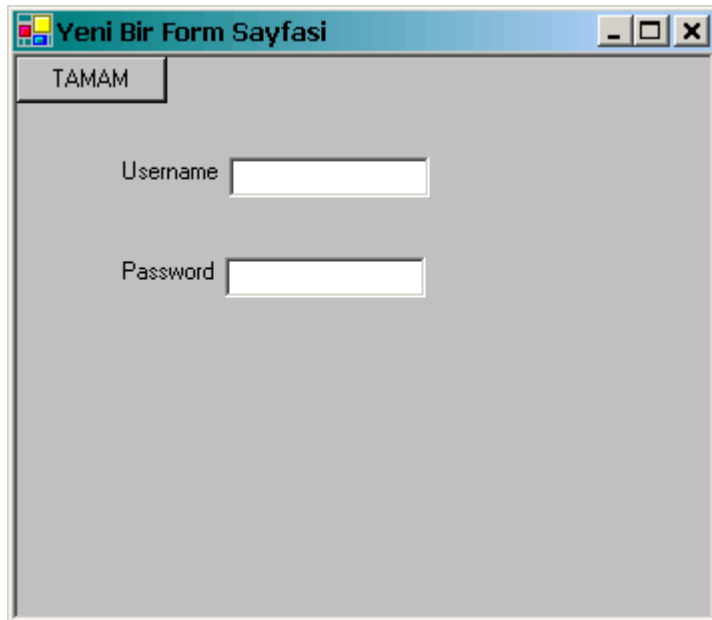
```
using System;
using System.Windows.Forms;
using System.Drawing;
public class BirForm:Form
{
    /* Kullanacağımız nesneler tanımlanıyor. İki adet Label nesnesi, iki adet TextBox nesnesi ve birde Button kontrolü. */
    private Label lbl1;
    private Label lbl2;
    private TextBox txtUsername;
    private TextBox txtPassword;
    private Button btnOK;
    public BirForm()
    {
        NesneleriAyarla();
    }
    private void NesneleriAyarla()
    {
        this.Text="Yeni Bir Form Sayfası"; /* this anahtar kelimesi oluşturulan Form nesnesini temsil eder.*/
        this.BackColor=Color.Silver; /* Formun arka plan rengini belirliyoruz. Color Enumaration'ınını kullanabilmek için Drawing sınıfının eklenmiş olması gereklidir. */
        this.StartPosition=FormStartPosition.CenterScreen; /* Form oluşturulduğunda ekranın ortasında görünmesi sağlanıyor. */
        this.FormBorderStyle=FormBorderStyle.Fixed3D; /* Formun border çizgileri 3 boyutlu ve sabit olarak belirleniyor. */
        /* Label nesnelerini oluşturuyor ve özelliklerini ayarlıyoruz. */
        lbl1=new Label();
        lbl2=new Label();
        lbl1.Text="Username";
        lbl1.Location=new Point(50,50); /* lbl1 nesnesini 50 birim sağa 50 birim aşağıya konumlandırıyoruz */
        lbl1.AutoSize=true; /* Label'ın boyutunun text uzunluğuna göre otomatik olarak ayarlanmasını sağlıyoruz. */
        lbl2.Text="Password";
        lbl2.Location=new Point(50,100); /* Bu kez 50 birim sağa, 100 birim aşağıya yerleştiriyoruz. */
        lbl2.AutoSize=true;
        /* TextBox nesnelerini oluşturuyor ve özelliklerini ayarlıyoruz. */
        txtUsername=new TextBox();
        txtPassword=new TextBox();
        txtUsername.Text="";
        txtUsername.Location=new Point(lbl1.PreferredWidth+50,50); /* Textbox nesnemizi lbl1 in uzunluğundan 50 birim fazla olacak şekilde sağa ve 50 birim aşağıya konumlandırıyoruz. */
    }
}
```

```

txtPassword.Text="";
txtPassword.Location=new Point(lbl2.PreferredWidth+50,100);
/* Button nesnemizi oluřturuyor ve özelliklerini belirliyoruz */
btnOK=new Button();
btnOK.Text="TAMAM";
btnOK.Location=new Point(0,0);
/* Buraya btnOK nesnesi için olay procedure tanımı eklenecek. */
/* řimdi kontrollerimizi Formumuza ekleyelim . Bunun için Form sınıfına ait Controls koleksiyonunu ve Add metodunu
kullanıyoruz. */
this.Controls.Add(lbl1);
this.Controls.Add(lbl2);
this.Controls.Add(txtUsername);
this.Controls.Add(txtPassword);
this.Controls.Add(btnOK);
this.Width=lbl1.PreferredWidth+txtUsername.Width+200; /* Son olarak formun genişliğini ve yüksekliğini ayarlıyoruz. */
this.Height=lbl1.PreferredWidth+lbl2.PreferredWidth+200;
}
/* Buraya btnOK için Click olay procedure kodları eklenecek. */
public static void Main()
{
    Application.Run(new BirForm());
}
}

```

Evet bu kodu derleyip çalıştırdığımızda aşağıdaki Form görüntüsünü elde etmiş oluruz.



Şekil 3. Form tasarımını geliştiriyoruz.

Şimdi işin en önemli kısımlarından birine geldi sıra. Oda olay güdümlü kodları yazmak. Yani kullanıcı etkilerine tepki vericek kodları yazmak. Kısaca Event-Handler. Kullanıcı bir eylem gerçekleştirdiğinde programın hangi tepkileri vereceğini belirtmek durumundayız. Bunun için şu syntax'ı kullanınız.

```
protected void metodunAdi(object sender, System.EventArgs e)
```

Burada metodumuzun hangi nesne için çalıştırılacağını sender anahtar kelimesi belirtir. Metod protected tanımlanır. Yani bulunduğu sınıf ve bulunduğu sınıftan türetilen sınıflarda kullanılabilir. Geri dönüş değeri yoktur. Bu genel formasyonla tanımlanan bir olay procedure'ünü nesne ile ilişkilendirmek için System.EventHandlerler delegeesi kullanılır.

```
nesneAdi.OlayinTanılayıcıBilgisi += new System.EventHandler(this.metodunAdi)
```

Yukarıdaki örneğimizde klasik örnek olarak, Button nesnemize tıklandığında çalıştırılması için bir Click olay procedure'ü ekleyelim.

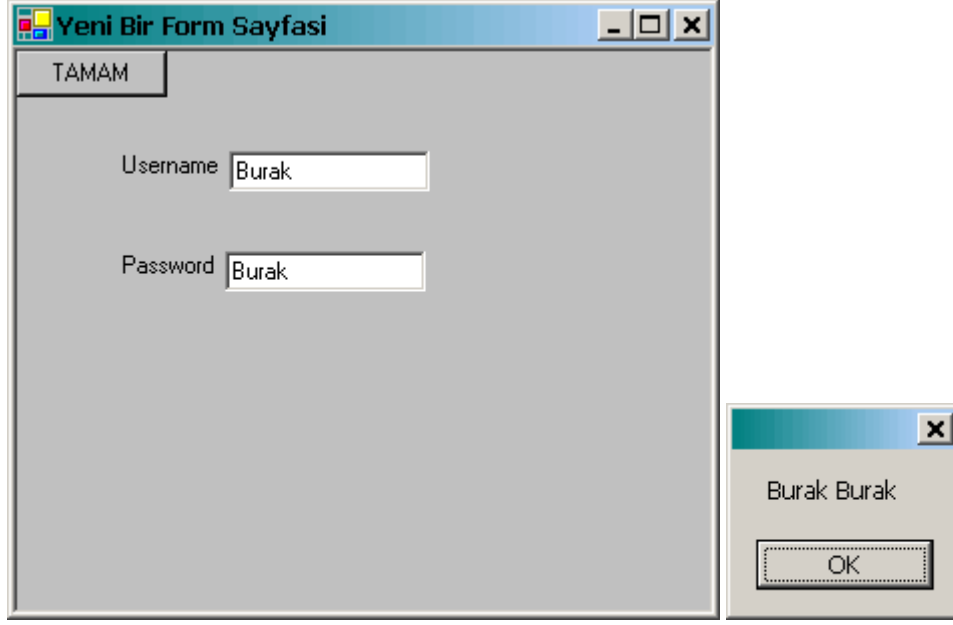
Şimdi /* Buraya btnOK nesnesi için olay procedure tanımı eklenecek */ yazan yere , aşağıdaki kod satırını ekliyoruz.

```
btnOK.Click += new System.EventHandler(this.btnOK_Tiklandi);
```

Bu satır ile btnOK nesnesine tıklandığında btnOK_Tiklandi isimli procedure'ün çalıştırılacağını belirtiyoruz. /* Buraya btnOK için Click olay procedure kodları eklenecek */ yazan yere ise olay procedure'ümüzün ve kodlarını ekliyoruz.

```
protected void btnOK_Tiklandi(object sender, System.EventArgs e)
{
    MessageBox.Show(txtUsername.Text + " " + txtPassword.Text);
}
```

Şimdi programı tekrar derleyip çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 4. Event-Handler sonucu.

Evet geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu ve huzurlu günler dilerim.

Params Anahtar Sözcüğünün Kullanımı

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, C# metodlarında önemli bir yere sahip olduğunu düşündüğüm params anahtar kelimesinin nasıl kullanıldığını incelemeye çalışacağız. Bildiğiniz gibi metodlara verileri parametre olarak aktarabiliyor ve bunları metod içerisinde işleyebiliyoruz. Ancak parametre olarak geçirilen veriler belli sayıda oluyor. Diyelimki sayısını bilmediğimiz bir eleman kümesini parametre olarak geçirmek istiyoruz. Bunu nasıl başarabiliriz? İşte params anahtar sözcüğü bu noktada devreye girmektedir. Hemen çok basit bir örnek ile konuya hızlı bir giriş yapalım.

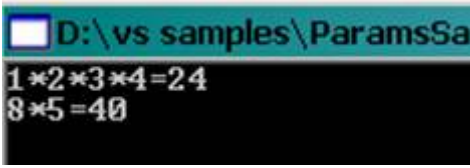
```
using System;
namespace ParamsSample1
{
    class Class1
    {
        /* burada Carpim isimli metodumuza, integer tipinde değerler geçirilmesini sağlıyoruz. params anahtarı bu metoda
        istediğimiz sayıda integer değer geçirebileceğimizi ifade ediyor*/
        public int Carpim(params int[] deger)
        {
            int sonuc=1;
            for(int i=0;i<deger.Length;++i) /*Metoda gönderilen elemanlar doğal olarak bir dizi oluştururlar. Bu dizideki elemanlara
            bir for döngüsü ile kolayca erişebiliriz. Dizinin eleman sayısını ise Length özelliği ile öğreniyoruz.*//
            {
```

```

        sonuc*=deger[i]; /* Burada metoda geçirilen integer deęerlerin birbirleri ile arpılmasını saęlıyoruz*/
    }
    return sonuc;
}
static void Main(string[] args)
{
    Class1 cl=new Class1();
    Console.WriteLine("1*2*3*4={0}",cl.Carpim(1,2,3,4)); /* Burada Carpim isimli metoda 4 integer deęer gnderdik.
Ařaęıdaki kodda ise 2 adet integer deęer gnderiyoruz.*/
    Console.WriteLine("8*5={0}",cl.Carpim(8,5));
    Console.ReadLine();
}
}
}

```

Bu rneęi alıřtıracak olursak, ařaęıdaki sonucu elde ederiz.



řekil 1. İlk Params rneęinin Sonucu

Peki derleyici bu iřlemi nasıl yapıyor birazda ondan bahsedelim. Carpim isimli metoda deęiřik sayılarda parametre gnderdięimizde, derleyici gnderilen parametre sayısı kadar boyuta sahip bir integer dizi oluřturur ve bu dizinin elemanlarına sırası ile (0 indexinden bařlayacak řekilde) gnderilen elemanları atar. Daha sonra aynı metodu bu eleman sayısı belli olan diziyi aktararak aęırır. cl.Carpim (8,5) satırını dřnelim; derleyici,

İlk adımda,

```
int[] dizi=new int[2] ile 2 elemanlı 1 dizi yaratır.
```

İkinci adımda,

```
dizi[0]=8
```

```
dizi[1]=5 řeklinde bu dizinin elemanlarını belirler.
```

Son adımda ise metodu tekrar aęırır.

```
cl.Carpim(dizi);
```

Bazı durumlarda parametre olarak geireceęimiz deęerler farklı veri tiplerine sahip olabilirler. Bu durumda params anahtar szcğünü, object tipinde bir dizi ile kullanırız. Hemen bir rnek ile grelim. Aynı rneęimize Goster isimli deęer dndrmeyen bir metod ekliyoruz. Bu metod kendisine aktarılan deęerleri console penceresine yazdırıyor.

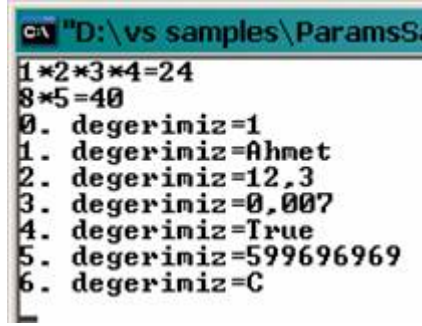
```

public void Goster(params object[] deger)
{
    for(int i=0;i<deger.Length;++i)
    {
        Console.WriteLine("{0}. değerimiz={1}",i,deger[i].ToString());
    }
    Console.ReadLine();
}

static void Main(string[] args)
{
    cl.Goster(1,"Ahmet",12.3F,0.007D,true,599696969,"C");
}

```

Görüldüğü gibi Goster isimli metodumuza değişik tiplerde (int,Float,Decimal,bool, String) parametreler gönderiyoruz. İşte sonuç;



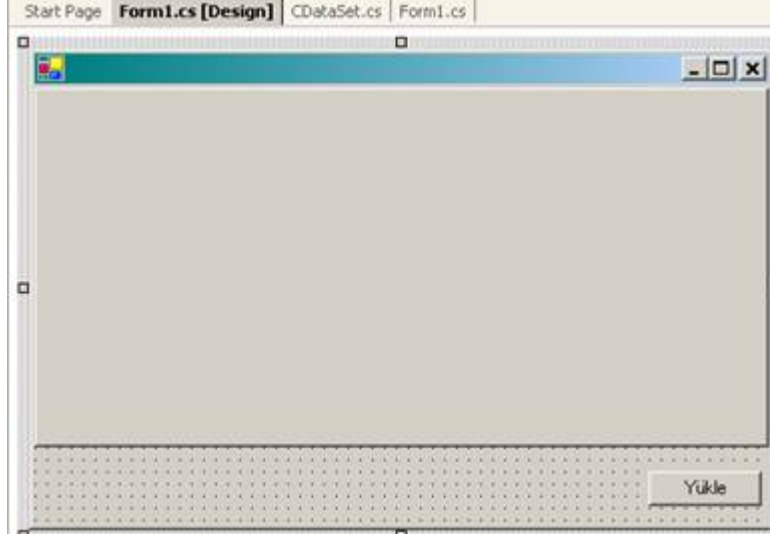
```

C:\> "D:\vs samples\ParamsS...
1*2*3*4=24
8*5=40
0. degeriniz=1
1. degeriniz=Ahmet
2. degeriniz=12,3
3. degeriniz=0,007
4. degeriniz=True
5. degeriniz=599696969
6. degeriniz=C

```

Şekil 2. params object[] kullanımı.

Şimdi dilerseniz daha işe yarar bir örnek üzerinde konuyu pekiştirmeye çalışalım. Örneğin değişik sayıda tabloyu bir dataset nesnesine yüklemek istiyoruz. Bunu yapacak bir metod yazalım ve kullanalım. Programımız, bir sql sunucusu üzerinde yer alan her hangibir database'e bağlanıp istenilen sayıdaki tabloyu ekranda programatik olarak oluşturulan dataGrid nesnelerine yükleyecek. Kodları inceledikçe örneğimizi daha iyi anlıyacaksınız.



Şekil 3. Form Tasarımımız

Uygulamamız bir Windows Application. Bir adet tabControl ve bir adet Button nesnesi içeriyor. Ayrıca params anahtar sözcüğünü kullanan CreateDataSet isimli metodumuzu içeren CDataSet isimli bir class'ımızda var. Bu class'a ait kodları yazarak işimize başlayalım.

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace CreateDataSet
{
    public class CDataSet
    {
        /* CreateDataSet isimli metod gönderilen baglantiAdi stringinin değerine göre bir SqlConnection nesnesi oluşturur. tabloAdi
        ile dataset nesnesine eklemek istediğimi tablo adlarını bu metoda göndermekteyiz. params anahtarı kullanıldığı için istediğimiz
        sayıda tablo adı gönderebiliriz. Elbette, geçerli bir Database ve geçerli tablo adları göndermeliyiz.*/
        public DataSet CreateDataSet(string baglantiAdi, params string[] tabloAdi)
        {
            string sqlSelect, conString;
            conString = "data source=localhost;initial catalog="+baglantiAdi+";integrated security=sspi"; /* Burada SqlConnection
            nesnesinin kullanacağı connectionString'i belirliyoruz.*/
            DataSet ds = new DataSet(); /* Tablolarımızı taşıyacak dataset nesnesini oluşturuyoruz*/
            SqlConnection con = new SqlConnection(conString); /*SqlConnection nesnemizi oluşturuyoruz*/
            SqlDataAdapter da; /* Bir SqlDataAdapter nesnesi belirtiyoruz ama henüz oluşturmuyoruz*/
            /*Bu döngü gönderdiğimiz tablo adlarını alarak bir Select sorgusu oluşturur ve SqlDataAdapter yardımıyla select sorgusu
            sonucu dönen tablo verilerini oluşturulan bir DataTable nesnesine yükler. Daha sonra ise bu DataTable nesnesi DataSet nesnemizin
            Tables koleksiyonuna eklenir. Bu işlem metoda gönderilen her tablo için yapılacaktır. Böylece döngü sona erdiğinde, DataSet
```

nesnemiz göndermiş olduğumuz tablo adlarına sahip DataTable nesnelerini içermiş olacaktır. */

```

        for(int i=0;i<tabloAdi.Length;++i)
        {
            sqlSelect="SELECT * FROM "+tabloAdi[i];
            da=new SqlDataAdapter(sqlSelect,con);
            DataTable dt=new DataTable(tabloAdi[i]);
            da.Fill(dt);
            ds.Tables.Add(dt);
        }
        return ds; /* Son olarak metod çağırıldığı yere DataSet nesnesini göndermektedir.*/
    }
    public CDataSet()
    {
    }
}

```

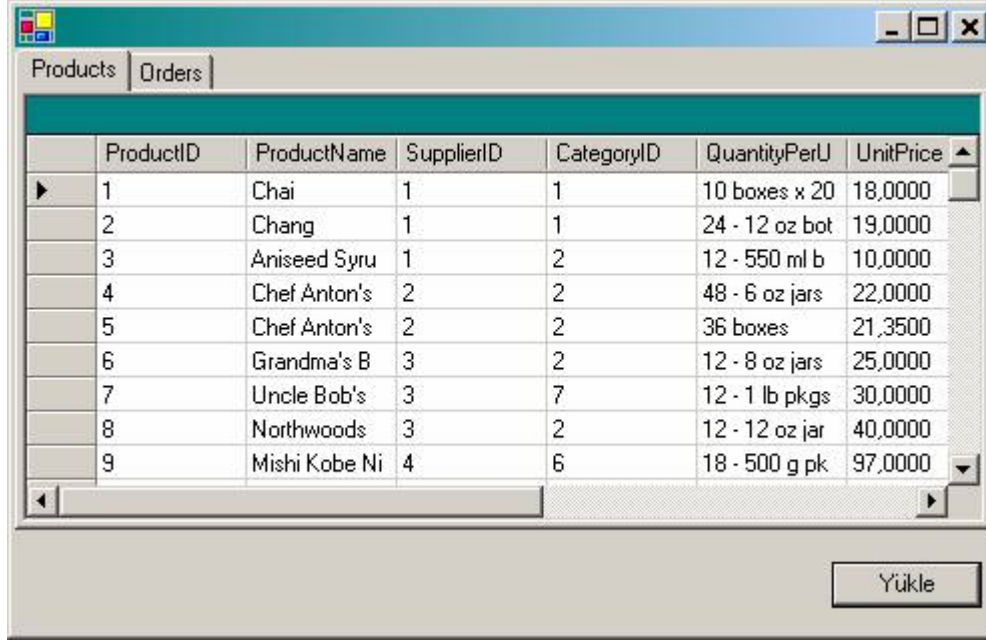
Şimdi ise btnYukle isimli butonumuzun kodlarını yazalım.

```

private void btnYukle_Click(object sender, System.EventArgs e)
{
    CDataSet c=new CDataSet();
    DataSet ds=new DataSet();
    ds=c.CreateDataSet("northwind","Products","Orders");
    for(int i=0;i<ds.Tables.Count;++i)
    {
        /* tabControl'umuza yeni bir tab page ekliyoruz.*/
        tabControl1.TabPages.Add(new System.Windows.Forms.TabPage(ds.Tables[i].TableName.ToString()));
        /* Oluşturulan bu tab page'e eklenmek üzere yeni bir datagrid oluşturuyoruz.*/
        DataGrid dg=new DataGrid();
        dg.Dock=DockStyle.Fill; /*datagrid tabpage'in tamamını kaplıyacak*/
        dg.DataSource=ds.Tables[i]; /* DataSource özelliği ile DataSet te i indexli tabloyu bağlıyoruz.*/
        tabControl1.TabPages[i].Controls.Add(dg); /* Oluşturduğumuz dataGrid nesnesini TabPage üstünde göstermek için Controls
        koleksiyonunun Add metodunu kullanıyoruz.*/
    }
}

```

Şimdi programımızı çalıştıralım. İşte sonuç;

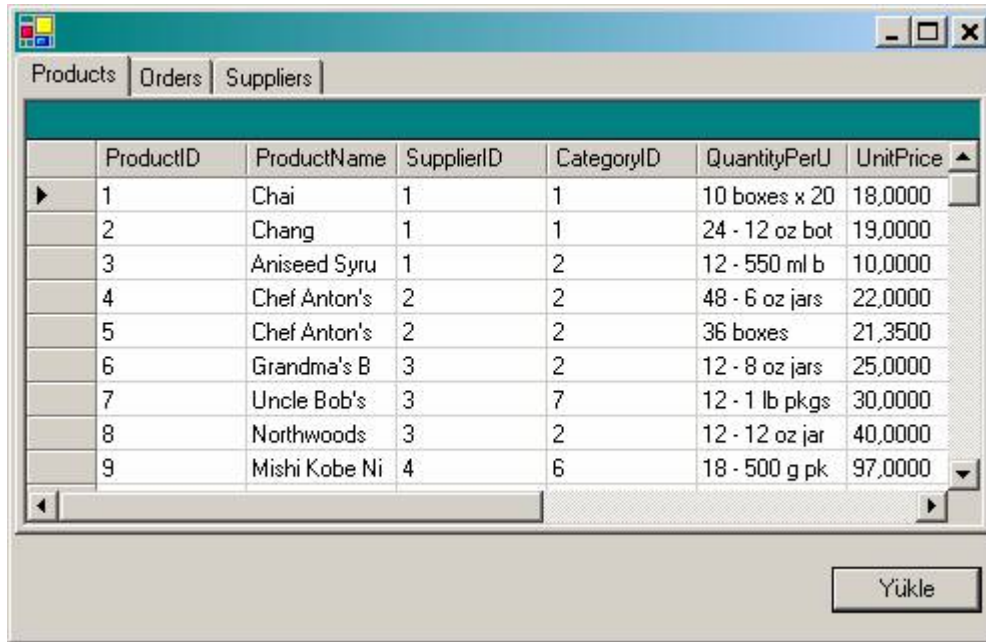


Şekil 4. Tabloların yüklenmesi.

Görüldüğü gibi iki tablomuzda yüklenmiştir. Burada tablo sayısını arttırabilir veya azaltabiliriz. Bunu params anahtar kelimesi mümkün kılmaktadır. Örneğin metodumuzu bu kez 3 tablo ile çağıralım;

```
ds=c.CreateDataSet("northwind","Products","Orders","Suppliers");
```

Bu durumda ekran görüntümüz Şekil 5 teki gibi olur.



Şekil 5. Bu kez 3 tablo gönderdik.

Umuyorumki params anahtar sözcüğü ile ilgili yeterince bilgi sahibi olmuşsunuzdur. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

DataSet ve WriteXml Metodunun Kullanımı

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, bir dataset nesnesinin içerdiği tabloların ve bu tablolardaki alanlara ait bilgilerin xml formatında nasıl yazdırıldığını göreceğiz. Örneğimiz son derece basit. Örnek uygulamamızda, Sql sunucusu üzerinde yer alan, Friends isimli database'den Kitaplar isimli tabloya ait verileri taşıyan bir dataset nesnesini kullanacağız. DataSet sınıfına ait WriteXml metodu dataset içerisinde yer alan bilgilerin bir xml dokümanına Schema bilgisi ile birlikte aktarılmasına imkan sağlamaktadır. Bu metoda ait 8 adet yapıcı(Constructor) metod bulunmakta olup biz örneğimizde;

Public void WriteXml(string dosyaadi, XmlWriteMode mod);

Yapıcısını kullanacağız. Burada yer alan ilk parametre xml içeriğini kaydedeceğimiz dosyanın tam yol adını taşımaktadır. İkinci parametre ise ;

XmlWriteMode.IgnoreSchema

XmlWriteMode.WriteSchema

XmlWriteMode.DiffGram

Değerlerinden birini alır. IgnoreSchema olarak belirtildiğinde, DataSet nesnesinin içerdiği veriler, Schema bilgileri olmadan (örneğin alan adları, veri tipleri, uzunlukları vb...) xml dokümanı haline getirilir. WriteSchema olması halinde ise, Schema bilgileri aynı xml dosyasının üst kısmına <xs:schema> ile </xs:schema> tagları arasında yazılır. DiffGram durumunda ise, veriler üzerinde yapılan değişikliklerin takip edilebilmesi amaçlanmıştır. Dilerseniz vakit kaybetmeden örnek uygulamamıza geçelim. Basit bir Console uygulaması oluşturacağız.

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace WriteXml
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
            SqlDataAdapter da=new SqlDataAdapter("Select Kategori,Adi,Yazar,BasimEvi From Kitaplar",conFriends);
            DataSet ds=new DataSet();
            conFriends.Open();
            da.Fill(ds);
            /*yukarıdaki adımlarda, Sql sunucumuz üzerinde yer alan Friends isimli database'e bir bağlantı açıyor, SqlDataAdapter
```

nesnemiz yardımıyla bu database içindeki Kitaplar isimli tablodan verileri alıyor ve bunları bir dataset nesnesine yüklüyoruz.*/

```
ds.WriteXml("D:\\Kitaplar.xml",XmlWriteMode.WriteSchema); /* Bu adımda ise, dataset nesnesini içerdiği verileri
Schema bilgisi ile birlikte Kitaplar.xml isimli xml dokumanına yazıyoruz. */
conFriends.Close(); // Bağlantımızla işlemiz bittiğinden kapatıyoruz.
}
}
}
```

Bu uygulamayı çalıştırdığımızda, D:\Kitaplar.xml isimli bir dosyanın oluştuğunu görürüz. Bu dosyayı açtığımızda ise aşağıda yer alan xml kodlarını elde ederiz. Gördüğümüz gibi verilerin yanında alanlara ait bilgilerde aynı xml dosyası içine yüklenmiştir.

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:Locale="tr-TR">
      <xs:complexType>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="Table">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Kategori" type="xs:string" minOccurs="0" />
                <xs:element name="Adi" type="xs:string" minOccurs="0" />
                <xs:element name="Yazar" type="xs:string" minOccurs="0" />
                <xs:element name="BasimEvi" type="xs:string" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <Table>
    <Kategori>Bilgisayar Programlama</Kategori>
    <Adi>Delphi 5'e Bakış</Adi>
    <Yazar>Ruhvar Barengi</Yazar>
    <BasimEvi>SECKIN</BasimEvi>
  </Table>
  <Table>
    <Kategori>Bilgisayar Programlama</Kategori>
    <Adi>Delphi 5 Uygulama Geliştirme Kılavuzu</Adi>
    <Yazar>Marco Cantu</Yazar>
    <BasimEvi>ALFA</BasimEvi>
  </Table>
</NewDataSet>
```

Kodu şimdide aşağıdaki gibi değiştirelim. IgnoreSchema seçimini kullanalım bu kezde.

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

```
using System.Data.SqlClient;
namespace WriteXml
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
            SqlDataAdapter da=new SqlDataAdapter("Select Kategori,Adi,Yazar,BasimEvi From Kitaplar",conFriends);
            DataSet ds=new DataSet();
            conFriends.Open();
            da.Fill(ds);
            ds.WriteXml("D:\\\\Kitaplar.xml",XmlWriteMode.IgnoreSchema);
            conFriends.Close();
        }
    }
}
```

Bu durumda, Kitaplar.xml dosyamızın içeriğine bakacak olursak schema bilgilerinin eklenmediğini sadece tablonun içerdiği verilerin yer aldığını görürüz.

```

<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Table>
    <Kategori>Bilgisayar Programlama</Kategori>
    <Adi>Delphi 5'e Bakış</Adi>
    <Yazar>Ruhvar Barengi</Yazar>
    <BasimEvi>SECKIN</BasimEvi>
  </Table>
  <Table>
    <Kategori>Bilgisayar Programlama</Kategori>
    <Adi>Delphi 5 Uygulama Geliştirme Kılavuzu</Adi>
    <Yazar>Marco Cantu</Yazar>
    <BasimEvi>ALFA</BasimEvi>
  </Table>
  <Table>
    <Kategori>Bilgisayar Programlama</Kategori>
    <Adi>Delphi 5 Kullanım Kılavuzu</Adi>
    <Yazar>Dr.Cahit Akın</Yazar>
    <BasimEvi>ALFA</BasimEvi>
  </Table>
  .
  .
  .
</NewDataSet>

```

Bir sonraki makalemizde görüşmek dileğiyle hepinizi mutlu günler dilerim.

Enumerators

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, kendi değer türlerimizi oluşturmanın yollarından birisi olan Enumerator'ları inceleyeceğiz. C# dilinde veri depolamak için kullanabileceğim temel veri türleri yanında kendi tanımlayabileceğimiz türlerde vardır. Bunlar Structs(Yapılar), Arrays(Diziler) ve Enumerators(Numaralandırıcılar) dır. Numaralandırıcılar, sınırlı sayıda değer içeren değişkenler yaratmamıza olanak sağlarlar. Burada bahsi geçen değişken değerleri bir grup oluştururlar ve sembolik bir adla temsil edilirler.

Numaralandırıcıları kullanma nedenlerimizden birisi verilere anlamlar yükleyerek, program içerisinde kolay okunabilmelerini ve anlaşılabilirliklerini sağlamaktır. Örneklerimizde bu konuyu çok daha iyi anlıyacaksınız. Bir Numaralandırıcı tanımlamak için aşağıdaki syntax kullanılır.

Kapsam belirteçleri enum numaralandırıcıAdi

```

{
    birinciUye,
    ikinciUye,
    ucuncuUye,
}

```

Kapsam belirteçleri protected,public,private,internal yada new değerini alır ve numaralandırıcının yaşayacağı kapsamı belirtir. Dikkat edilecek olursa, elemanlara herhangi bir değer ataması yapılmamıştır. Nitekim Numaralandırıcıların özelliğidir bu. İlk eleman 0 değerine sahip olmak üzere diğer elemanlar 1 ve 2 değerlerini sahip olacak şekilde belirlenirler. Dolayısıyla programın herhangi bir yerinde bu numaralandırıcıya ait elemana ulaştığımızda, bu elemanın index değerine erişmiş oluruz. Gördüğümüz gibi numaralandırıcı kullanmak okunurluğu arttırmaktadır.Dilersek numaralandırıcı elemanlarının 0 indexinden değil de her hangibir değerden başlamasını sağlayabilir ve hatta diğer elemanlarada farklı index değerleri atayabiliriz. Basit bir Numaralandırıcı örneği ile konuyu daha iyi anlamaya çalışalım.

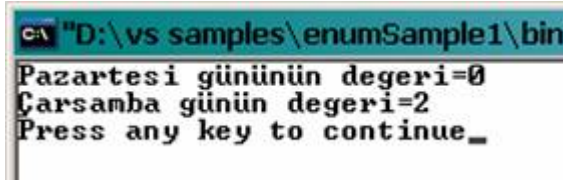
```
using System;
namespace enumSample1
{
    class Class1
    {
        /* Haftanın günlerini temsil edecek bir numaralandırıcı tipi oluşturuyoruz. Pazartesi 0 index değerine sahip iken Pazar 6 index
        değerine sahip olacaktır.*/
        enum Gunler
        {
            Pazartesi,
            Sali,
            Carsamba,
            Persembe,
            Cuma,
            Cumartesi,
            Pazar
        }
        static void Main(string[] args)
        {
            Console.WriteLine("Pazartesi gününün değeri={0}",(int)Gunler.Pazartesi);
            Console.WriteLine("Çarşamba günün değeri={0}",(int)Gunler.Carsamba);
        }
    }
}
```

Burada Gunler. Yazdıktan sonar VS.NET 'in intellisense özelliği sayesinde, numaralandırıcının sahip olduğu değerler kolayca ulaşabiliriz.



Şekil 1. Intellisense sağolsun.

Programı çalıştıracak olursak aşağıdaki ekran görüntüsünü elde ederiz.

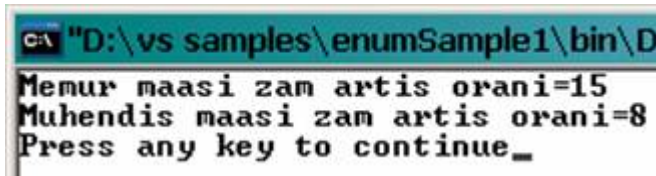


Şekil 2. İlk Örnek

Şimdi başka bir örnek geliştirelim. Bu kez numaralandırıcının değerleri farklı olsun.

```
enum Artis
{
    Memur=15,
    Isci=10,
    Muhendis=8,
    Doktor=17,
    Asker=12
}

static void Main(string[] args)
{
    Console.WriteLine("Memur maaşı zam artış oranı={0}",(int)Artis.Memur);
    Console.WriteLine("Muhendis maaşı zam artış oranı= {0}",(int)Artis.Muhendis);
}
```



Şekil 3. İkinci Örneğin Sonucu

Dikkat edicek olursak, numaralandırıcıları program içinde kullanırken, açık olarak (explicit) bir dönüşüm yapmaktayız. Şu ana kadar numaralandırıcı elemanlarına integer değerler atadık. Ama dilersek Long tipinden değerde atayabiliriz. Fakat bu durumda enum ' ın değeri türünde belirtmemiz gerekmektedir.Örneğin,

```
enum Sinirlar:long
{
    EnBuyuk=458796452135L,
    EnKucuk=255L
}
static void Main(string[] args)
{
    Console.WriteLine("En üst sınır={0}",(long)Sinirlar.EnBuyuk);
    Console.WriteLine("Muhendis maaşı zam artış oranı={0}",(long)Sinirlar.EnKucuk);
}
```

Görüldüğü gibi Sinirlar isimli numaralandırıcı long tipinde belirtilmiştir. Bu sayede numaralandırıcı elemanlarına long veri tipinde değerler atanabilmektedir. Dikkat edilecek bir diğer noktada, bu elemanlara ait değerleri kullanırken, long tipine dönüştürme yapılmasıdır. Bir numaralandırıcı varsayılan olarak integer tiptedir. Bu nedenle integer değerleri olan bir numaralandırıcı tanımlanırken int olarak belirtilmesine gerek yoktur. Şimdi daha çok işe yarar bir örnek geliştirmeye çalışalım. Uygulamamız son derece basit bir forma sahip ve bir kaç satır koddan oluşuyor. Amacımız numaralandırıcı kullanmanın programcı açısından işleri daha da kolaylaştırıyor olması. Uygulamamız bir Windows Application. Form tasarımı aşağıdaki gibi olacak.

Şekil 4. Form Tasarımımız.

Form yüklenirken Şehir Kodlarının yer aldığı comboBox kontrolümüz otomatik olarak numaralandırıcının yardımıyla doldurulacak. İşte program kodları.

```
public enum AlanKodu
{
    Anadolu=216,
    Avrupa=212,
    Ankara=312,
    Izmir=412
}
```

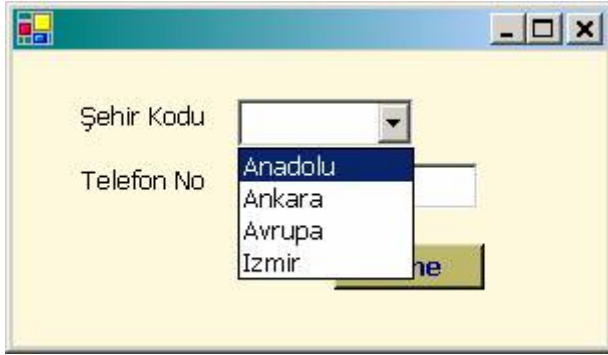


```

}
private void Form1_Load(object sender, System.EventArgs e)
{
    comboBox1.Items.Add(AlanKodu.Anadolu);
    comboBox1.Items.Add(AlanKodu.Ankara);
    comboBox1.Items.Add(AlanKodu.Avrupa);
    comboBox1.Items.Add(AlanKodu.Izmir);
}

```

İşte sonuç,



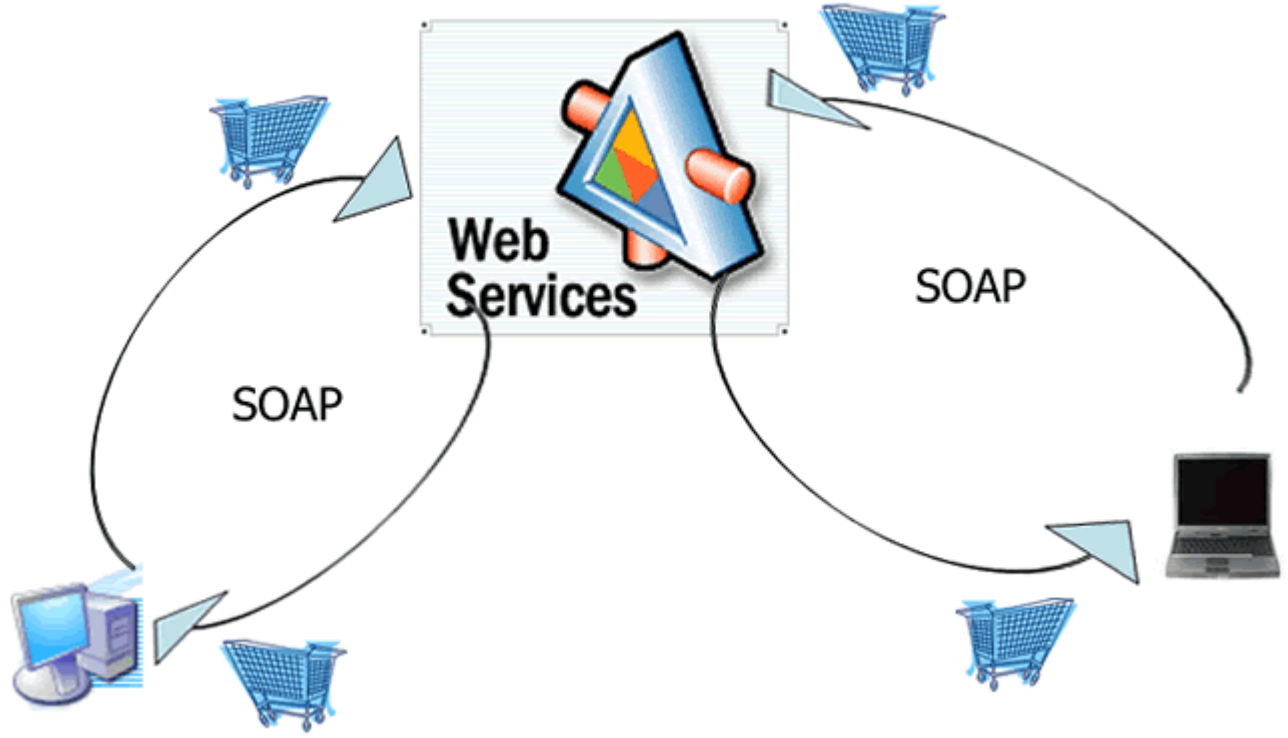
Şekil 5 . Sonuç .

Aslında bu comboBox kontrolünü başka şekillerde de alan kodları ile yükleyebiliriz . Bunu yapmanın sayısız yolu var. Burada asıl dikkat etmemiz gereken nokta numaralandırıcı sayesinde bu sayısal kodlarla kafamızı karıştırmak yerine daha bilinen isimler ile aynı sonuca ulaşmamızdır. Geldik bir makalemizin daha sonuna. İlerleyen makalelerimizde bu kez yine kendi değer tiplerimizi nasıl yaratabileceğimize struct kavramı ile devam edeceğiz. Hepinize mutlu günler dilerim.

Basit Bir Web Service Uygulaması

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde web servislerinin nasıl kullanıldığını göreceğiz. Her zaman olduğu gibi konuyu açıklayıcı basit bir örnek üzerinde çalışacağız. Öncelikle web servisi nedir, ne işe yarar bunu açıklamaya çalışalım. Web servisi, internet üzerinden erişilebilen, her türlü platform ile bağlantı kurabileceğimiz, geriye sonuç döndüren(döndürmeye) fonksiyonelliklere ve hizmetlere sahip olan bir uygulama parçasıdır. Aşağıdaki şekil ile konuyu zihnimizde daha kolay canlandırabiliriz.



Şekil 1 . Web Servislerinin yapısı

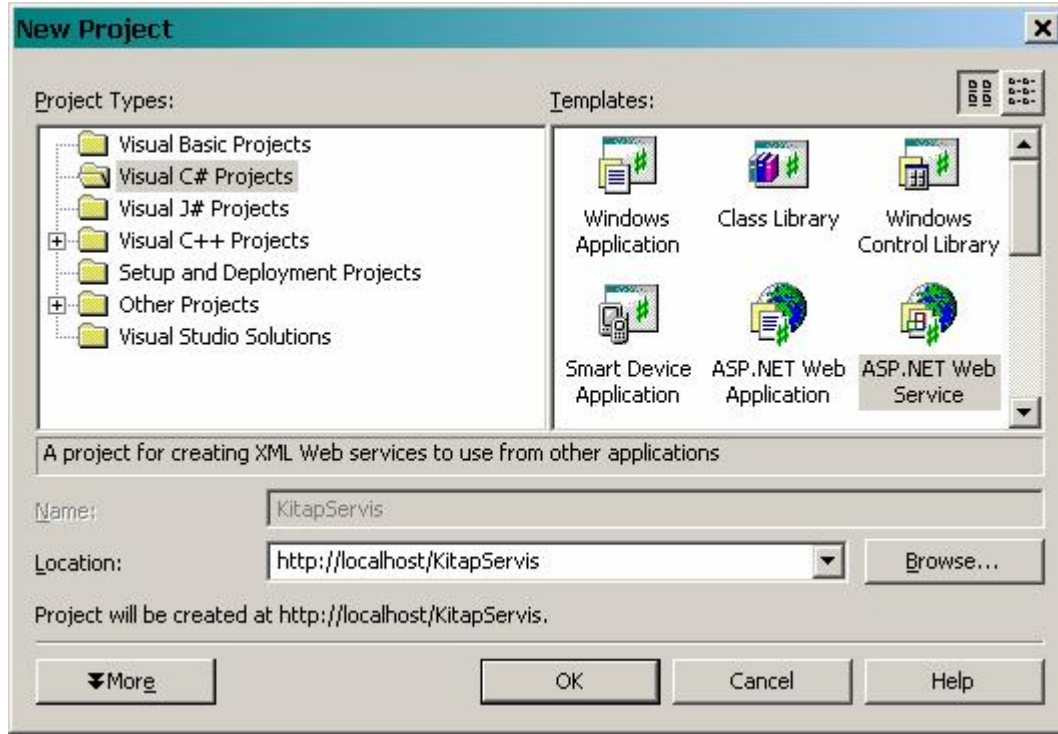
İşekildende görüldüğü gibi SOAP isminde bir yapıdan bahsediyoruz. SOAP Simple Object Access Protocol anlamına gelen XML tabanlı bir haberleşme teknolojisidir. Web servisi ile bu web servisinin kullanan uygulamalar arasındaki iletişimi sağlar. Dolayısıyla, web servisleri ve bu servisleri kullanan uygulamaların birbirlerini anlayabilmesini sağlar. Web servislerinden uygulamalara giden veri paketleri ve uygulamalardan web servislerine giden veri paketleri bu standart protokolü kullanmaktadır. Web servislerinin yazıldığı dil ile bunları kullanan uygulamaların (bir windows application, bir web application vb...) yazıldığı diller farklı olabilir. SOAP aradaki iletişim standartlaştırdığından farklı diller sorun yaratmaz. Tabi bunu sağlayan en büyük etken SOAP'ın XML tabanlı teknolojisidir.

Bir web servis uygulaması yaratıldığında, bu web servisi kullanacak olan uygulamaların web servisinin nerede olduğunu öncelikle bilmesi gerekir. Bunu sağlayan ise web proxy'lerdir. Bir web proxy yazmak çoğunlukla kafa karıştırıcı kodlardan oluşmaktadır. Ancak VS.Net gibi bir ortamda bunu hazırlamakta oldukça kolaydır. Uygulamaya, web servisin yeri Web Proxy dosyası ile bildirildikten sonra, uygulamamızdan bu web servis üzerindeki metodlara kolayca erişebiliriz.

Şimdi, basit bir web servis uygulaması geliştireceğiz. Öncelikle web servis'imizi yazacağız. Daha sonra ise, bu web servisini kullanacağımız bir windows application geliştireceğiz. Normalde birde web proxy uygulaması geliştirmemiz gerekiyor. Ancak bu işi VS.NET'e bırakacağız.

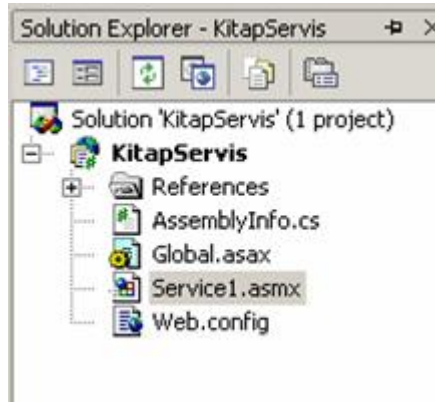
Basit olması açısından web servis'imiz, bulunduğu sunucu üzerindeki bir sql sunucusunda yer alan bir veritabanından, bir tabloya ait veri kümesini DataSet nesnesi olarak, servisi çağıran uygulamaya geçirecek. Geliştirdiğim uygulamam aynı makine üzerindeki bir web servisini kullanıyor. Öncelikle web servisimizi oluşturalım. Bunun için New Project'ten ASP.NET Web Service'ı seçiyoruz.

Görüldüğü gibi uygulama otomatik olarak internet information server'ın kurulu olduğu sanal web sunucusunda oluşturuluyor. Dolayısıyla oluşturulan bu web servis'e bir browser yardımıyla erişebiliriz.



Şekil 2. Web Service

Bu işlemi yaptığımız takdirde Service1.asmx isimli bir dosya içeren bir web servis uygulamasının oluşturulduğunu görürüz. Web servislerinin dosya uzantısı asmx dir. Bu uzanti çalışma zamanında veya bir tarayıcı penceresinde bu dosyanın bir web servis olduğunu belirtir.



Şekil 3. Web servisleri asmx uzantısına sahiptir.

ITo Switch Code Window ile kod penceresine geçtiğimizde aşağıdaki kodları görürüz. (Burada, Service1 ismi SrvKitap ile değiştirilmiş aynı zamanda Constructor'un adı ve Solution Explorer'da yer alan dosya adıda SrvKitap yapılmıştır.)

```
using System;
```

```
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
namespace KitapServis
{
    /// <summary>
    /// Summary description for Service1.
    /// </summary>
    public class SrvKitap : System.Web.Services.WebService
    {
        public SrvKitap()
        {
            //CODEGEN: This call is required by the ASP.NET Web Services Designer
            InitializeComponent();
        }

        #region Component Designer generated code
        //Required by the Web Services Designer
        private IContainer components = null;
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
        }
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if(disposing && components != null)
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

```

    }
    #endregion

    // WEB SERVICE EXAMPLE
    // The HelloWorld() example service returns the string Hello World
    // To build, uncomment the following lines then save and build the project
    // To test this web service, press F5

    // [WebMethod]
    // public string HelloWorld()
    // {
    //     return "Hello World";
    // }
}

```

Kodları kısaca inceleyecek olursak, web servislere ait özellikleri kullanabilmek için System.Web.Services namespace'inin eklendiğini, SrvKitap isimli sınıfın bir web servisin niteliklerine sahip olması için, System.Web.Services sınıfından türetildiğini görürüz. Ayrıca yorum satırı olarak belirtilen yerde VS.NET bize hazır olarak HelloWorld isimli bir web metodu sunmaktadır. Bu metodun başındada dikkat edicek olursanız [WebMethod] satırı yer alıyor. Bu anahtar sözcük, izleyen metodun bir web servis metodu olduğunu belirtmektedir. Şimdi biz kendi web servis metodumuzu buraya ekleyelim.

```

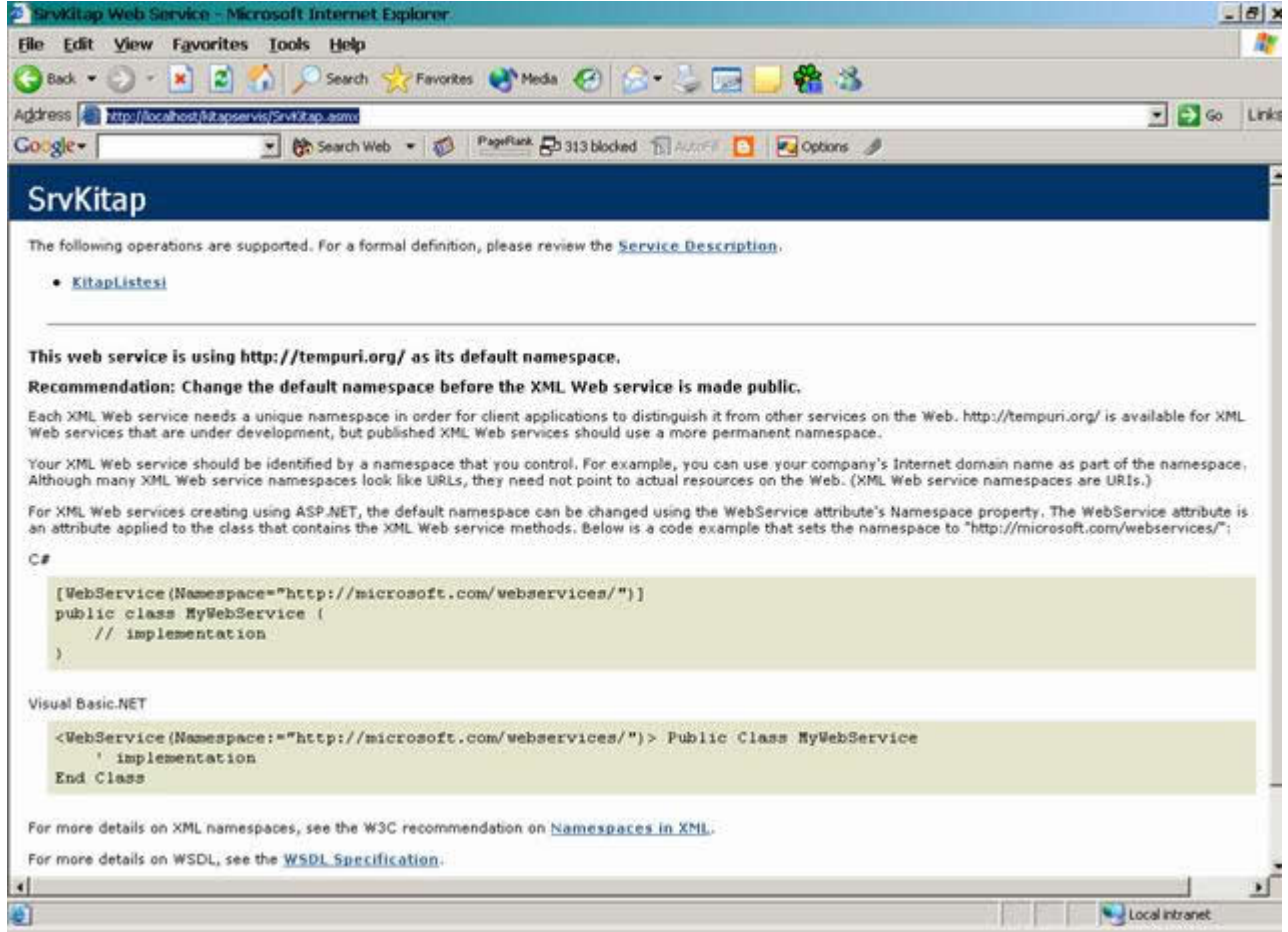
[WebMethod]
public DataSet KitapListesi()
{
    SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
    SqlDataAdapter da=new SqlDataAdapter("Select Adi,Fiyat From Kitaplar",conFriends);
    DataSet ds=new DataSet();
    da.Fill(ds);
    return ds;
}

```

Eklediğimiz bu web metod sadece Sql sunucusu üzerinde yer alan Friends isimli veritabanına bağlanmakta ve burada Kitaplar isimli tablodan Adi ve Fiyat bilgilerini alarak, sonuç kümesini bir DataSet içine aktarmaktadır. Sonra metod bu DataSet'I geri döndürmektedir.Şimdi uygulamamızı derleyelim ve Internet Explorer'ı açarak adres satırına şunu girelim

<http://localhost/kitapservis/SrvKitap.aspx>

bu durumda, aşağıdaki ekran görüntüsünü alırsınız.



Şekil 4. Internet Explorer penceresinde SrvKitap.asmx'in görüntüsü.

SrvKitap isimli web servisimiz burada görülmektedir. Şimdi KitapListesi adlı linke tıklarsak (ki bu link oluşturduğumuz KitapListesi adlı web servis metoduna işaret etmektedir)



Şekil 5. Invoke

Invoke butonuna basarak web servisimizi test edebiliriz. Bunun sonucunda metod çalıştırılır ve sonuçlar xml olarak gösterilir.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
= <DataSet xmlns="http://tempuri.org/">
```

```
= <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```

xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:Locale="tr-TR">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="Table">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Adi" type="xs:string" minOccurs="0" />
              <xs:element name="Fiyat" type="xs:decimal" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>

<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
  <NewDataSet xmlns="">
    <Table diffgr:id="Table1" msdata:rowOrder="0">
      <Adi>Delphi 5'e Bakış</Adi>
      <Fiyat>100.0000</Fiyat>
    </Table>
    <Table diffgr:id="Table2" msdata:rowOrder="1">
      <Adi>Delphi 5 Uygulama Geliştirme Kılavuzu</Adi>
      <Fiyat>250.0000</Fiyat>
    </Table>
    <Table diffgr:id="Table3" msdata:rowOrder="2">
      <Adi>Delphi 5 Kullanım Kılavuzu</Adi>
      <Fiyat>50.0000</Fiyat>
    </Table>
    <Table diffgr:id="Table4" msdata:rowOrder="3">
      <Adi>Microsoft Visual Basic 6.0 Geliştirmek Ustalaşma Dizisi</Adi>
      <Fiyat>75.0000</Fiyat>
    </Table>
    <Table diffgr:id="Table5" msdata:rowOrder="4">
      <Adi>Visual Basic 6 Temel Başlangıç Kılavuzu</Adi>
      <Fiyat>80.0000</Fiyat>
    </Table>
  </NewDataSet>
</diffgram>

```

```

<Table diffgr:id="Table6" msdata:rowOrder="5">
  <Adi>Microsoft Visual Basic 6 Temel Kullanım Kılavuzu Herkes İçin!</Adi>
  <Fiyat>15.0000</Fiyat>
</Table>
<Table diffgr:id="Table7" msdata:rowOrder="6">
  <Adi>ASP ile E-Ticaret Programcılığı</Adi>
  <Fiyat>25.0000</Fiyat>
</Table>
<Table diffgr:id="Table8" msdata:rowOrder="7">
  <Adi>ASP 3.0 Active Server Pages Web Programcılığı Temel Başlangıç Kılavuzu</Adi>
  <Fiyat>150.0000</Fiyat>
</Table>
.
.
.
</NewDataSet>
</diffgr:diffgram>
</DataSet>

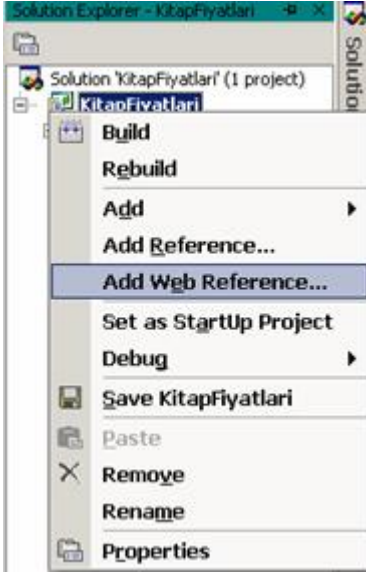
```

Sıra geldi bu web servisini kullanacak olan uygulamamızı yazmaya. Örneğin bir Windows Uygulamasından bu servise erişelim. Yeni bir Windows Application oluşturalım ve Formumuzu aşağıdakine benzer bir şekilde tasarlayalım.



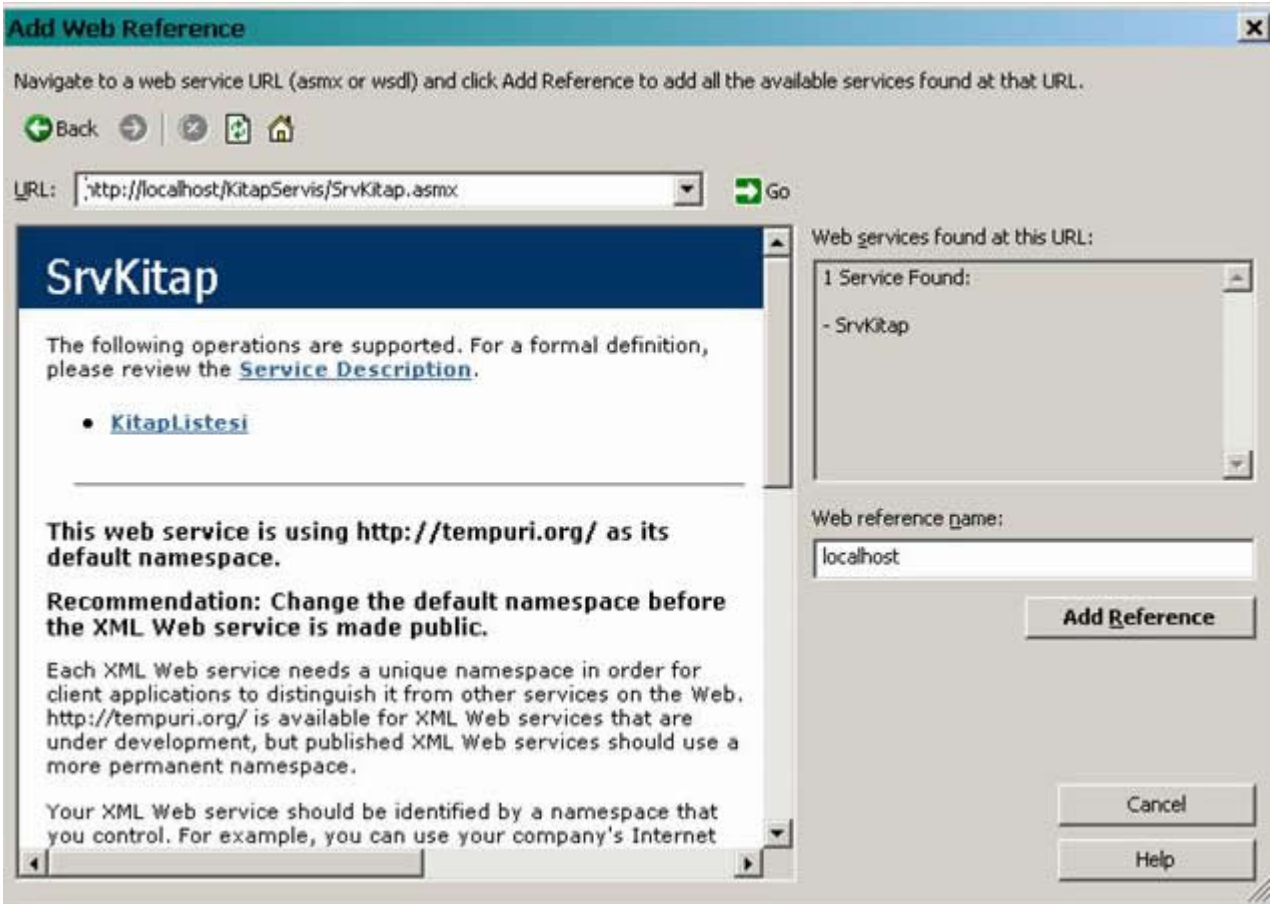
Şekil 6 Form Tasarımı.

Formumuz bir adet datagrid nesnesi ve bir adet button nesnesi içeriyor. Button nesnemize tıkladığımızda, web servisimizdeki KitapListesi adlı metodumuzu çağırarak ve dataGrid'imizi bu metoddan dönen dataSet'e bağlayacağız. Ancak öncelikle uygulamamıza, web servisin yerini belirtmeliyiz ki onu kullanabilelim. Bunun için Solution Explorer penceresinde Add Web Reference seçimi yapıyoruz.



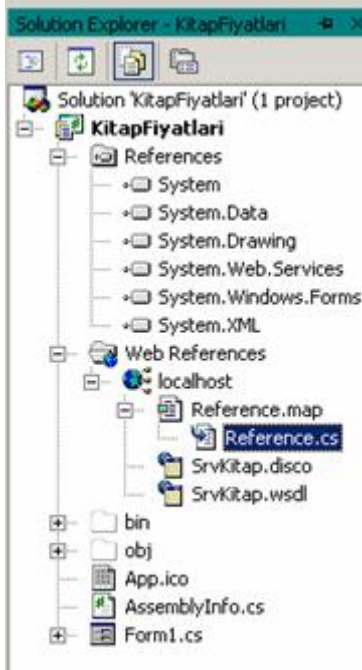
Şekil 7. Uygulamaya bir Web Reference eklemek.

Karşımıza gelen pencerede adres satırına `http://localhost/KitapServis/SrvKitap.asmx` (yani web servisimizin adresi) yazıyor ve Go tuşuna basıyoruz. Web service bulunduğunda bu bize 1 Service Found ifadesi ile belirtiliyor.



Şekil 8. Servisin eklenmesi.

Add Reference diyerek servisimizi uygulamamıza ekliyoruz. Bu durumda Solution Explorer'dan uygulamamızın içerdiği dosyalara baktığımızda yeni dosyaların eklendiğini görüyoruz.

*Şekil 9. Reference.cs adlı dosyasına dikkat.*

Burada Reference.cs isimli dosya işte bizim yazmaktan çekindiğimi Web Proxy dosyasının ta kendisi oluyor. Bu dosyanın kodları ise aşağıdaki gibidir.

```
//-----
// <autogenerated>
// This code was generated by a tool.
// Runtime Version: 1.1.4322.573
//
// Changes to this file may cause incorrect behavior and will be lost if
// the code is regenerated.
// </autogenerated>
//-----

//
// This source code was auto-generated by Microsoft.VSDesigner, Version 1.1.4322.573.
//
namespace KitapFiyatlari.localhost
{
```

```

using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Web.Services;
/// <remarks/>
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="SrvKitapSoap", Namespace="http://tempuri.org/")]
public class SrvKitap : System.Web.Services.Protocols.SoapHttpClientProtocol
{
    /// <remarks/>
    public SrvKitap()
    {
        this.Url = "http://localhost/KitapServis/SrvKitap.asmx";
    }
    /// <remarks/>
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/KitapListesi",
RequestNamespace="http://tempuri.org/", ResponseNamespace="http://tempuri.org/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public System.Data.DataSet KitapListesi()
    {
        object[] results = this.Invoke("KitapListesi", new object[0]);
        return ((System.Data.DataSet)(results[0]));
    }
    /// <remarks/>
    public System.IAsyncResult BeginKitapListesi(System.AsyncCallback callback, object asyncState)
    {
        return this.BeginInvoke("KitapListesi", new object[0], callback, asyncState);
    }
    /// <remarks/>
    public System.Data.DataSet EndKitapListesi(System.IAsyncResult asyncResult)
    {
        object[] results = this.EndInvoke(asyncResult);
        return ((System.Data.DataSet)(results[0]));
    }
}

```

```
}

```

Karmaşık olduğu gözlenen bu kodların açıklamasını ilerleyen makalerimde işeyeceğim. Artık web servisimizi uygulamamıza eklediğimize göre, bunu kullanmaya ne dersiniz. İşte button nesnemize tıklandığında çalıştırılacak kodlar.

```
private void button1_Click(object sender, System.EventArgs e)
{
    localhost.SrvKitap srv=new localhost.SrvKitap(); /* Servisimizi kullanabilmek için bu servisten bir örnek nesne yaratıyoruz*/
    DataSet ds=new DataSet();
    ds=srv.KitapListesi(); /* Servisteki KitapListesi isimli metodumuzu çağırıyoruz.*
    dataGrid1.DataSource=ds;
}
```



Şekil 10. Sonuç.

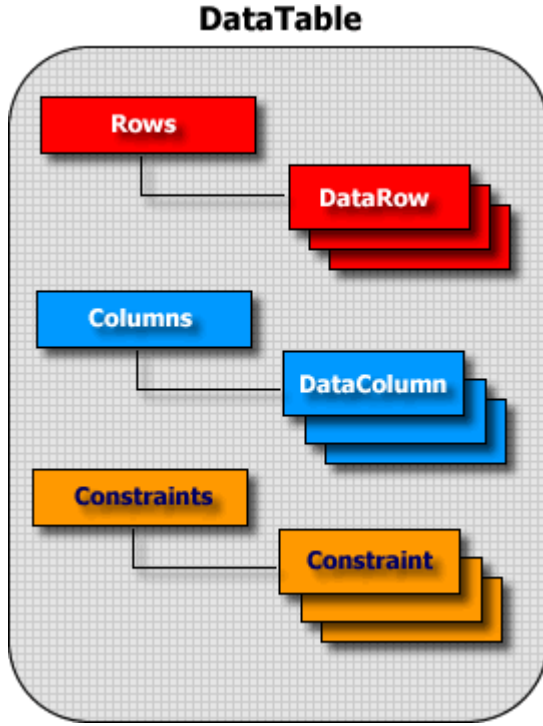
Evet geldik bir makalemizin daha sonuna . Bir sonraki makalemizde görüşmek dileğiyle hepinizle mutlu günler dilerim.

DataTable Sınıfını Kullanarak Programatik Olarak Tablolar Oluşturmak-1

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde bağlantısız katmanın önemli bir sınıfı olan DataTable nesnesini bir açıdan incelemeye çalışacağız. Bilindiği gibi DataTable sınıfından türetilen bir nesne, bir tabloyu ve elemanlarını bellekte temsil etmek için kullanılmaktadır. DataTable sınıfı bellekte temsil ettiği tablolara ait olan satırları Rows koleksiyonuna ait DataRow nesneleri ile temsil ederken, tabloun alanlarını ise, Columns koleksiyonuna ait DataColumn nesneleri ile temsil etmektedir. Örnek uygulamamızda bu sınıf nesnelerini detaylı olarak

kullanacağız. Diğer yandan DataTable sınıfı bir tabloya ilişkin kısıtların yer aldığı Constraints koleksiyonuna ait Constraint nesnelerinedee sahiptir. DataTable sınıfının ve üye elemanlarını aşağıdaki şekilde daha kolayca canlandırabiliriz.

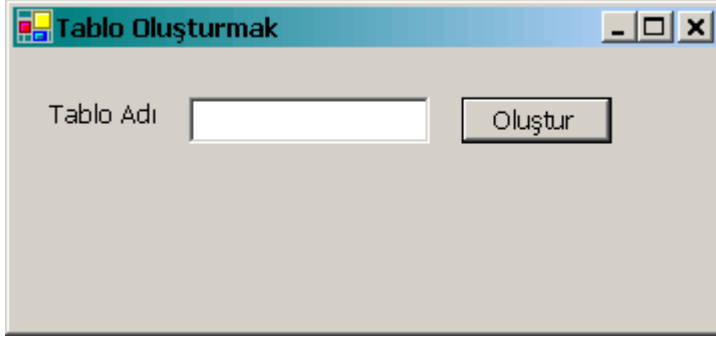


Şekil 1 DataTable mimarisi

Geliştireceğimiz uygulamada, bizim belirlediğimiz alanlardan oluşan bir tabloyu bellekte oluşturmaya çalışacağız. Öncelikle DataTable nesnesi ile bir tablo oluşturmak için aşağıdaki adımları takip etmeliyiz.

- 1 – Bir DataTable nesnesi oluşturup DataTable’ın bellekte temsil edeceği tablo için bir isim belirlenir.
- 2 – Tablomuzun içereceği alanların isimleri, veri türleri belirlenerek birer DataRow nesnesi şeklinde, DataTable nesnesinin Columns koleksiyonuna eklenir.
- 3 – Tablomuz için bir primary key alanı belirlenir.

Bu adımların ardından tablomuz bellekte oluşturulmuş olacaktır. Bu noktadan sonra bu tablo üzerinde dilediğimiz işlemleri yapabiliriz. Kayıt ekleyebilir, silebilir, sorgulayabiliriz. Ama tabiki programı kapattığımızda bellekteki tablonun yerinde yeller esiyor olacaktır. Ama üzülmenin ilerliyen makalelerimizde SQL-DMO komutları yardımıyla programımız içinden bir sql sunucusu üzerinde veritabanı oluşturacak ve tablomuzu buraya ekleyeceğiz.Şimdi dilerseniz birinci adımdan itibaren bu işlerin nasıl yapıldığını minik örnekler ile inceleyelim ve daha sonrada asıl uygulamamızı yazalım. Öncelikle işe tablomuzu bellekte temsil edecek datatable nesnesi oluşturarak başlayalım. Aşağıdaki küçük uygulamayı oluşturalım.

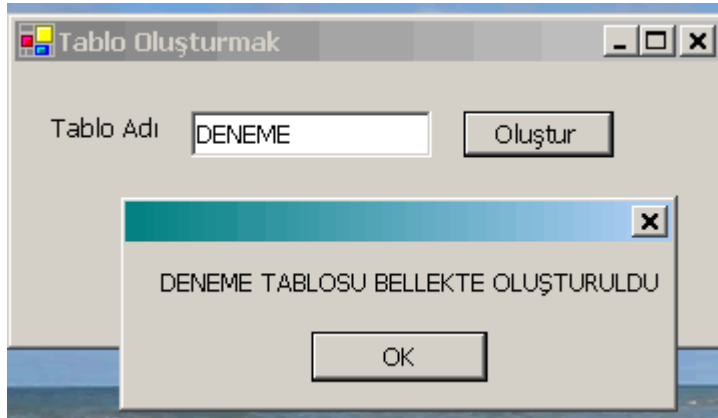


Şekil 2. Formun ilk hali

Ve kodlar,

```
private void btnTabloOlustur_Click(object sender, System.EventArgs e)
{
    /* Bir tabloyu bellekte temsil edecek bir datatable nesnesi oluşturarak işe başlıyoruz. Tablomuza txtTabloAdi isimli
    TextBox'a girilene değeri isim olarak veriyoruz */
    DataTable dt=new DataTable(txtTabloAdi.Text);
    MessageBox.Show(dt.TableName.ToString()+" TABLOSU BELLEKTE OLUŞTURULDU");
}
```

Şimdi programımızı çalıştıralım ve tablo ismi olarak DENEME diyelim. İşte sonuç,



Şekil 3. DataTable nesnesi oluşturuldu.

Şimdi ise tablomuza nasıl field(alan) ekleyeceğimize bakalım. Önceden bahsettiğimiz gibi tablonun alanları aslında DataTable sınıfının Columns koleksiyonuna ait birer DataColumn nesnesidir. Dolayısıyla öncelikle bir DataRow nesnesi oluşturup bu nesneyi ilgili DataTable'ın Columns koleksiyonuna eklememiz gerekmektedir. Alanın ismi dışında tabiki veri türünde belirtmeliyiz. Bu veri türlerini belirtirken Type.GetType syntaxı kullanılır. Formumuzu biraz değiştirelim. Kullanıcı belirlediği isimde ve türdeki alanı, tabloya ekleyebilecek olsun. Söylemek isterimki bu uygulamada hiç bir kontrol mekanizması uygulanmamış ve hataların önüne geçilmeye çalışılmamıştır. Nitekim amacımız DataTable ile bir tablonun nasıl oluşturulacağına dair basit bir örnek vermektir. Formumuzu aşağıdaki gibi değiştirelim. Kullanıcı bir tablo adı girip oluşturduktan sonra istediği alanları ekleyecek ve bu bilgiler listbox nesnemizde kullanıcıya ayrıca gösterilecek.

Şekil 4. Formumuzun yeni hali.

Şimdide kodlarımızı görelim.

DataTable dt; /* DataTable nesnemizi uygulama boyunca kullanabilmek için tüm metodların dışında tanımladık. */

private void btnTabloOlustur_Click(object sender, System.EventArgs e)

{

dt=new DataTable(txtTabloAdi.Text);

MessageBox.Show(dt.TableName.ToString()+" TABLOSU BELLEKTE OLUŞTURULDU");

lblTabloAdi.Text=dt.TableName.ToString(); /* TableName özelliği DataTable nesnesinin bellekte temsil ettiği tablonun adını vermektedir.*/

}

private void btnAlanEkle_Click(object sender, System.EventArgs e)

{

/*Önce yeni alanımız için bir DataColumn nesnesi oluşturulur*/

DataColumn dc=new DataColumn();

/*Şimdi ise DataTable nesnemizin Columns koleksiyonuna oluşturulan alanımızı ekliyoruz. İlk parametre, alanın adını temsil ederken, ikinci parametre ise alanın veri türünü belirtmektedir. Add metodu bu özellikleri ile oluşturulan DataColumn nesnesini dataTable'a ekler. Bu sayede tabloda alanımız oluşturulmuş olur.*/

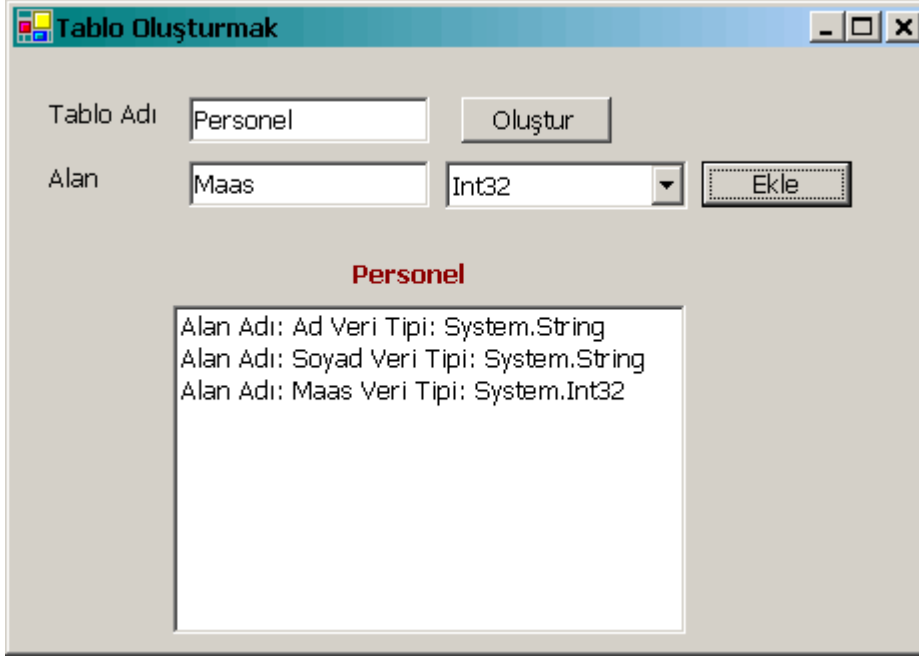
dt.Columns.Add(txtAlanAdi.Text,Type.GetType("System."+cmbAlanTuru.Text));

lstAlanlar.Items.Add("Alan Adı: "+dt.Columns[txtAlanAdi.Text].ColumnName.ToString()+" Veri Tipi: "+dt.Columns[txtAlanAdi.Text].DataType.ToString());

/* ColumnName özelliği ile eklenen alanın adını, DataType özelliği ile de bu alanın veri türünü öğreniyoruz.*/

}

Uygulamamızı deneyelim.



Şekil 5. Alanlarımızda tabloya ekleyelim.

Alanlara veri türlerini aktarırken kullanabileceğimiz diğer örnek değerler aşağıdaki gibidir; bunlar listbox kontrolüne desing time(tasarım zamanında) eklenmiştir.

System.Int32
 System.Int16
 System.Int64
 System.Byte
 System.Char
 System.Single
 System.Decimal
 System.Double
 Gibi...

Şimdi de seçtiğimiz bir alanı primary key olarak belirleyelim. Unique (benzersiz) integer değerler alıcak bir alan olsun bu ve 1000 den başlayarak 1'er 1'er otomatik olarak artsın. Bildiğini ID alanlarından bahsediyorum. Bunu kullanıcıya sormadan otomatik olarak biz yaratalım ve konudan fazlaca uzaklaşmayalım. Sadece btnTabloOlustur'un kodlarına ekleme yapıyoruz.

```
private void btnTabloOlustur_Click(object sender, System.EventArgs e)
{
    dt=new DataTable(txtTabloAdi.Text);
    MessageBox.Show(dt.TableName.ToString()+" TABLOSU BELLEKTE OLUŞTURULDU");
}
```


lblTabloAdi.Text=dt.TableName.ToString(); /* TableName özelliği DataTable nesnesinin bellekte temsil ettiği tablonun adını vermektedir.*/

/* Tablo oluşturulduğunda otomatik olarak bir ID alanı ekleyelim. Bu alan benzersiz bir alan olacak yani içerdiği veriler tekrar etmeyecek. 1 den başlayarak birer birer otomatik olarak artacak. Aynı zamanda primary key olacak. Primary key olduğu zaman arama gibi işlemleri yaparken bu alanı kullanacağız.*/

```
DataColumn dcID=new
DataColumn(); /* DataColumn nesnesi oluşturulur.*/
dcID.ColumnName="ID"; /* Alanın adı veriliyor.*/
dcID.DataType=Type.GetType("System.Int32");/* Alanınveritipi belirleniyor*/
dcID.Unique=true;/* Alanın içerdiği verilerin tekrar etmeyeceğisöyleniyor.*/
dcID.AutoIncrement=true;/* Alanın değerlerinin otomatik olarakartacağı söyleniyor.*/
dcID.AutoIncrementSeed=1;/* İlk değeri 1 olacak.*/
dcID.AutoIncrementStep=1;/* Alanın değerleri 1'er artacak.*/
dt.Columns.Add(dcID);/* Yukarıda özellikleri belirtilen ID alanıtablomuza ekleniyor. */
/* Aşağıdaki kod satırları ile ID isimli alanıPrimary Key olarak belirliyoruz. */
DataColumn[] anahtarlar=new DataColumn[1];
anahtarlar[0]=dt.Columns["ID"];
dt.PrimaryKey=anahtarlar;
lstAlanlar.Items.Add(dt.Columns["ID"].ColumnName.ToString()+"Primary Key");
}
```

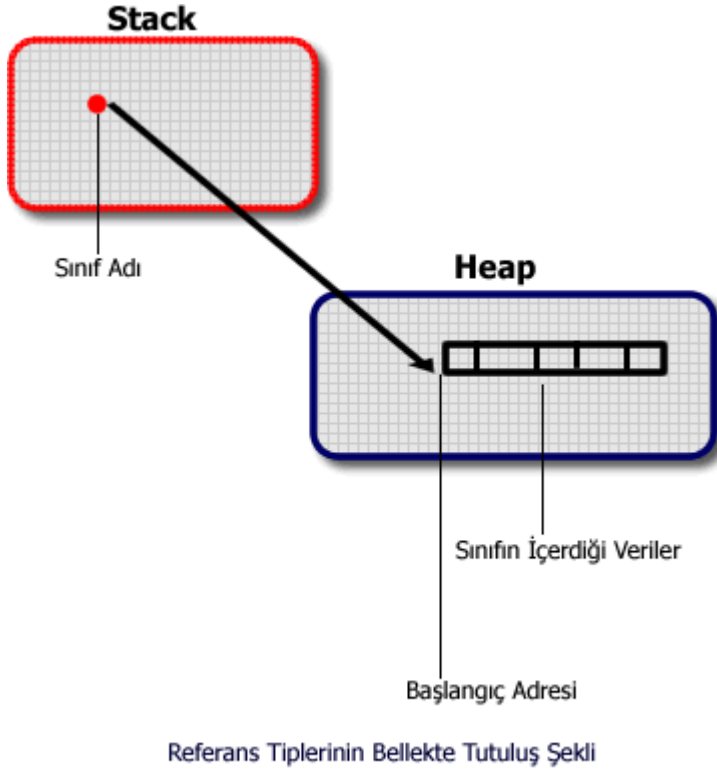
Geldik bir makalemizin daha sonuna. Bir sonrakimakalemizde oluşturduğumuz bu tabloya nasıl veri ekleneceğini göreceğimiz çokkisa bir uygulama yazacağız. Hepinizi mutlu günler dilerim.

Struct (Yapı) Kavramı ve Class (Sınıf) ile Struct (Yapı) Arasındaki Farklar

Değerli Okurlarım, Merhabalar.

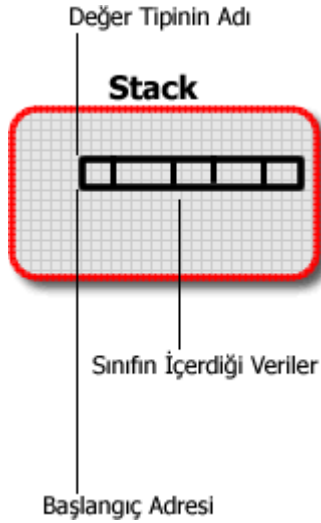
Bugünkü makalemizde struct kavramını incelemeye çalışacağız. Hatırlayacağınız gibi, kendi tanımladığımız veri türlerinden birisi olan Numaralandırıcıları (Enumerators) görmüştük. Benzer şekilde diğer bir veri tipide struct (yapı) lardır.Yapılar, sınıflar ile büyük benzerliklik gösterirler. Sınıf gibi tanımlanırlar. Hatta sınıflar gibi, özellikler,metodlar,veriler, yapıcılar vb... içerebilirler. Buna karşın sınıflar ile yapılar arasında çok önemli farklılıklar vardır. I

Herşeyden önce en önemli fark, yapıların değer türü olması ve sınıfların referans türü olmasıdır. Sınıflar referans türünden oldukları için, bellekte tutuluş biçimleri değer türlerine göre daha farklıdır. Referans tiplerinin sahip olduğu veriler belleğin öbek(heap) adı verilen tarafında tutulurken, referansın adı stack(yığın) da tutulur ve öbekteki verilerin bulunduğu adresi işaret eder. Ancak değer türleri belleğin stack denilen kısmında tutulurlar. Aşağıdaki şekil ile konuyu daha net canlandırabiliriz. I



Şekil 1. Referans Tipleri

Aşağıdaki şekilde ise değer tiplerinin bellekte nasıl tutulduğunu görüyorsunuz. I



Şekil 2. Değer Tipleri

İşte sınıflar ile yapılar arasındaki en büyük fark budur. Peki bu farkın bize sağladığı getiriler nelerdir? Ne zaman yapı ne zaman sınıf kullanmalıyız? Özellikle metodlara veriler aktarırken bu verileri sınıf içerisinde tanımladığımızda, tüm veriler metoda aktarılacağını sadece bu verilerin öbekteki başlangıç adresi aktarılır ve ilgili parametrenin de bu adresteki verilere işaret etmesi sağlanmış olur. Böylece büyük boyutlu verileri stack'ta kopyalayarak gereksiz miktarda bellek harcanmasının önüne geçilmiş olunur. Ancak küçük

boyutlarda veriler ile çalışırken bu verileri sınıflar içerisinde kullandığımızda bu kezde gereksiz yere bellek kullanıldığı öbek şişer ve performans düşer. Bu konudaki uzman görüş 16 byte'tan küçük veriler için yapıların kullanılması, 16 byte'tan büyük veriler için ise sınıfların kullanılmasıdır. I

Diğer taraftan yapılar ile sınıflar arasında başka farklılıklarda vardır. Örneğin bir yapı için varsayılan yapıcı metod (default constructor) yazamayız. Derleyici hatası alırız. Ancak bu değişik sayıda parametreler alan yapıcılar yazmamızı engellemez. Oysaki sınıflarda istersek sınıfın varsayılan yapıcı metodunu kendimiz yazabilmekteyiz. I

Bir yapı içerisinde yer alan constructor metod(lar) içinde tanımlanmış olduğumuz alanlara başlangıç değerlerini atamak zorundayız. Oysaki bir sınıftaki constructor(lar) içinde kullanılan alanlara başlangıç değerlerini atamaz isek, derleyici bizim yerimize sayısal değerlere 0, boolean değerlere false vb... gibi başlangıç değerlerini kendisi otomatik olarak yapar. Ancak derleyici aynı işi yapılarda yapmaz. Bu nedenle bir yapı içinde kullandığımız constructor(lar)daki tanımlanmış olduğumuz alanlara mutlaka ilk değerlerini vermemiz gerekir. Ancak yinede dikkat edilmesi gereken bir nokta vardır. Eğer yapı örneğini varsayılan yapılandırıcı ile oluşturursak bu durumda derleyici yapı içinde kullanılan alanlara ilk değerleri atanmamış ise kendisi ilk değerleri atar. Unutmayın, parametrelili constructorlarda her bir alan için başlangıç değerlerini bizim vermemiz gerekmektedir. Örneğin, aşağıdaki Console uygulamasını inceleyelim. I

```
using System;
namespace StructSample1
{
    struct Zaman
    {
        private int saat,dakika,saniye;
        private string kosucuAdi;
        public string Kosucu
        {
            get
            {
                return kosucuAdi;
            }
            set
            {
                kosucuAdi=value;
            }
        }
        public int Saat
        {
            get
            {
                return saat;
            }
        }
    }
}
```

```
        set
        {
            saat=value;
        }
    }
    public int Dakika
    {
        get
        {
            return dakika;
        }
        set
        {
            dakika=value;
        }
    }
    public int Saniye
    {
        get
        {
            return saniye;
        }
        set
        {
            saniye=value;
        }
    }
}
class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
        Zaman z;
        Console.WriteLine("Koşucu:"+z.Kosucu);
        Console.WriteLine("Saat:"+z.Saat.ToString());
        Console.WriteLine("Dakika:"+z.Dakika.ToString());
        Console.WriteLine("Saniye:"+z.Saniye.ToString());
    }
}
```

```

    }
}

```

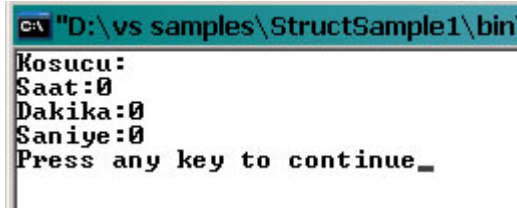
Yukarıdaki kod derlenmeyecektir. Nitekim derleyici **"Use of unassigned local variable 'z'"** hatası ile z yapısı için ilk değerlerin atanmadığını bize söyleyecektir. Ancak z isimli Zaman yapı türünü new anahtarı ile tanımlarsak durum değişir. I

```

Zaman z;
Satırı yerine
Zaman z=new Zaman();

```

yazalım .Bu durumda kod derlenir. Uygulama çalıştığında aşağıdaki ekran görüntüsü ile karşılaşırız. Görüldüğü gibi z isimli yapı örneğini new yapılandırıcısı ile tanımladığımızda, derleyici bu yapı içindeki özelliklere ilk değerleri kendi atamıştır. Kosucu isimli özellik için null, diğer integer özellikler için ise 0.



Şekil 3.New yapılandırıcısı ile ilk değer ataması.

Yine önemli bir farkta yapılarda türetme yapamıyacağımızdır. Bilindiği gibi bir sınıf oluşturduğumuzda bunu başka bir temel sınıftan kalıtım yolu ile türetebilmekteyiz ki inheritance olarak geçen bu kavramı ilerleyen makalelerimizde işleyeceğiz. Ancak bir yapıyı başka bir yapıyı temel alarak türetemeyiz. Şimdi yukarıda verdiğimiz örnekteki yapıdan başka bir yapı türetmeye çalışalım.

```

struct yeni:Zaman
{
}

```

satırlarını kodumuza ekleyelim.Bu durumda uygulamayı derlemeye çalıştığımızda aşağıdaki hata mesajını alırız.

'Zaman' : type in interface list is not an interface

Bu belirgin farklılıkları inceledikten sonra dilerseniz örneklerimiz ile konuyu pekiştirmeye çalışalım.

```

using System;
namespace StructSample1
{
    struct Zaman
    {
        private int saat,dakika,saniye;
        private string kosucuAdi;
        /* Yapı için parametrelili bir constructor metod tanımladık. Yapı içinde yer alan kosucuAdi,saat,dakika,saniye alanlarına ilk

```

değerlerin atandığına dikkat edelim. Bunları atamassak derleyici hatası alırız. */

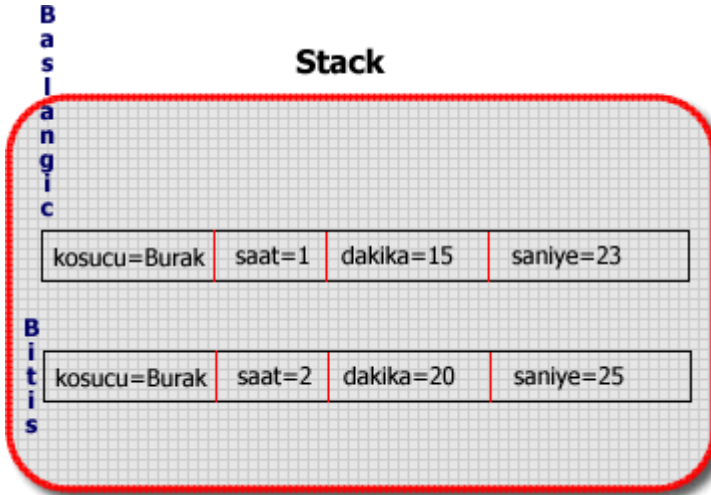
```
public Zaman(string k,int s,int d,int sn)
{
    kosucuAdi=k;
    saat=s;
    dakika=d;
    saniye=sn;
}
/* Bir dizi özellik tanımlayarak private olarak tanımladığımız asıl alanların kullanımını kolaylaştırıyoruz. */
public string Kosucu
{
    get
    {
        return kosucuAdi;
    }
    set
    {
        kosucuAdi=value;
    }
}
public int Saat
{
    get
    {
        return saat;
    }
    set
    {
        saat=value;
    }
}
public int Dakika
{
    get
    {
        return dakika;
    }
    set
    {
```

```

        dakika=value;
    }
}
public int Saniye
{
    get
    {
        return saniye;
    }
    set
    {
        saniye=value;
    }
}
}
class Class1
{
    static void Main(string[] args)
    {

```

/* Zaman yapısı içinde kendi yazdığımız parametrelili constructorlar ile Zaman yapısı örnekleri oluşturuyoruz. Yaptığımız bu tanımlamaların ardından belleğin stack bölgesinde derhal 4 adet değişken oluşturulur ve değerleri atanır. Yani kosucuAdi,saat,dakika,saniye isimli private olarak tanımladığımız alanlar bellekte stack bölgesinde oluşturulur ve atadığımız değerleri alırlar. Bu oluşan veri dizisinin adıda Zaman yapısı tipinde olan Baslangic ve Bitis değişkenleridir. */



```
Zaman Baslangic=new Zaman("Burak",1,15,23);
```

```
Zaman Bitis=new Zaman("Burak",2,20,25);
```

/* Zaman yapısı içinde tanımladığımız özelliklere erişip işlem yapıyoruz. Burada elbette zamanları birbirinden bu şekilde çıkarmak matematiksel olarak bir cinayet. Ancak amacımız yapıların kullanımını anlamak. Bu satırlarda yapı içindeki özelliklerimizin değerlerine erişiyor ve bunların değerleri ile sembolik işlemler yapıyoruz */

```
int saatFark=Bitis.Saat-Baslangic.Saat;
int dakikaFark=Bitis.Dakika-Baslangic.Dakika;
int saniyeFark=Bitis.Saniye-Baslangic.Saniye;
Console.WriteLine("Fark {0} saat, {1} dakika, {2} saniye",saatFark,dakikaFark,saniyeFark);
}
}
}
```

Bir sonraki makalemizde görüşmek dileğiyle. Hepinize mutlu günler dilerim

DataTable Sınıfını Kullanarak Programatik Olarak Tablolar Oluşturmak-2

Değerli Okurlarım, Merhabalar.

Hatırlayacağınız gibi yazı dizimizin ilk bölümünde, DataTable sınıfını kullanarak bellekte bir tablonun ve bu tabloya ait alanların nasıl yaratıldığını işlemiştik. Bugünkü makalemizde oluşturmuş olduğumuz bu tabloya kayıtlar ekleyeceğiz ve sonra bu DataTable nesnesini bir DataSet'e aktarıp içerisindeki verileri bir DataGridView kontrolünde göstereceğiz. Bir dataTable nesnesinin bellekte temsil ettiği tabloya yeni satırlar başka bir deyişle kayıtlar eklemek için, DataRow sınıfından nesneleri kullanacağız. Dilerseniz hiç vakit kaybetmeden uygulamamıza başlayalım. İlk örneğimizin devamı niteliğinde olacak bu çalışmamızda kullanıcının oluşturduğu tablodaki alan sayısı kadar textBox nesnesinide label nesneleri ile birlikte programatik olarak oluşturacağız. İşte programımızın kodları,

DataTable dt; /* DataTable nesnemizi uygulama boyunca kullanabilmek için tüm metodların dışında tanımladık. */

private void btnTabloOlustur_Click(object sender, System.EventArgs e)

{

dt=new DataTable(txtTabloAdi.Text);

MessageBox.Show(dt.TableName.ToString()+" TABLOSU BELLEKTE OLUSTURULDU");

lblTabloAdi.Text=dt.TableName.ToString(); /* TableName özelliği DataTable nesnesinin bellekte temsil ettiği tablonun adını vermektedir.*/

/*Tablo olusturulduğunda otomatik olarak bir ID alanı ekleyelim. Bu alan benzersiz bir alan olacak yani içerdiği veriler tekrar etmeyecek. 1 den başlayarak birer birer otomatik olarak artacak. Aynı zamanda primary key olacak. Primary key olduğu zaman arama gibi işlemleri yaparken bu alanı kullanacağız.*/

DataColumn dcID=new DataColumn(); /* DataColumn nesnesi oluşturulur.*/

dcID.ColumnName="ID"; /* Alanın adı veriliyor*/

dcID.DataType=Type.GetType("System.Int32"); /* Alanın veritipi belirleniyor*/

dcID.Unique=true; /* Alanın içerdiği verilerin tekrar etmeyeceği söyleniyor.*/

dcID.AutoIncrement=true; /* Alanın değerlerinin otomatik olarak artacağı söyleniyor.*/

dcID.AutoIncrementSeed=1; /* İlk değeri 1 olacak.*/


```

dcID.AutoIncrementStep=1;/* Alanin degerleri 1'er artacak.*/
dt.Columns.Add(dcID);/* Yukarida özellikleri belirtilen ID alanı tablomuza ekleniyor. */
/* Asagidaki kod satirlari ile ID isimli alanı Primary Key olarak belirliyoruz. */
DataColumn[] anahtarlar=new DataColumn[1];
anahtarlar[0]=dt.Columns["ID"];
dt.PrimaryKey=anahtarlar;
IstAlanlar.Items.Add(dt.Columns["ID"].ColumnName.ToString()+" Primary Key");
}
private void btnAlanEkle_Click(object sender, System.EventArgs e)
{
    /*Önce yeni alanımız için bir DataColumn nesnesi olusturulur*/
    DataColumn dc=new DataColumn();
    /*Simdi ise DataTable nesnemizin Columns koleksiyonuna olusturulan alanimizi ekliyoruz. İlk parametre, alanin adini temsil
    ederken, ikinci parametre ise alanin veri türünü belirtmektedir. Add metodu bu özellikleri ile olusturulan DataColumn nesnesini
    dataTable'a ekler. Bu sayede tabloda alanımız olusturulmus olur.*/
    dt.Columns.Add(txtAlanAdi.Text,Type.GetType("System."+cmbAlanTuru.Text));
    IstAlanlar.Items.Add("Alan Adi: "+dt.Columns[txtAlanAdi.Text].ColumnName.ToString()+" Veri Tipi:
    "+dt.Columns[txtAlanAdi.Text].DataType.ToString());
    /* ColumnName özelliği ile eklenen alanin adini, DataType özelliği ilede bu alanin veri türünü öğreniyoruz.*/
}
public void KontrolOlustur(string alanAdi,int index)
{
    /* Aşağıdaki kodları asıl konumuzdan uzaklaşmadan,açıklamak istiyorum. DataTable'daki her bir alan için bir TextBox nesnesi ve
    Label nesnesi olusturulup formumuza ekleniyor. Top özelliğinin ayarlanışına dikkatinizi çekmek isterin. Top özelliğini bu metoda
    gönderdiğimiz index paramteresi ile çarpıyoruz. Böylece, o sırada hangi indexli datacolumn nesnesinde isek ilgili kontrolün formun
    üst noktasından olan uzaklığı o kadar KAT artıyor. Elbette en önemli özellik kontrollerin adlarının verilemsi. Tabi her bir kontrolü
    this.Controls.Add syntax'ı formumuza eklemeyi unutmuyoruz. */
    System.Windows.Forms.TextBox txtAlan=new System.Windows.Forms.TextBox();
    txtAlan.Text="";
    txtAlan.Left=275;
    txtAlan.Top=30*index;
    txtAlan.Name="txt"+alanAdi.ToString();
    txtAlan.Width=100;
    txtAlan.Height=30;
    this.Controls.Add(txtAlan);
    System.Windows.Forms.Label lblAlan=new System.Windows.Forms.Label();
    lblAlan.Text=alanAdi.ToUpper();
    lblAlan.Left=200;
    lblAlan.Top=30*index;

```

```

        lblAlan.AutoSize=true;
        this.Controls.Add(lblAlan);
    }
    private void btnKontrol_Click(object sender, System.EventArgs e)
    {
        /* Aşağıdaki döngü ile dataTable nesnemizdeki DataColumn sayısı kadar dönecek bir döngü oluşturuyoruz. Yanlık ID alanımız (0
        indexli alan) değeri otomatik olarak atandığı için bu kontrolü oluşturmamıza gerek yok. Bu nedenle döngümüz 1 den başlıyor.
        Döngü her bir DataColumn nesnesi için, bu alanın adını parametre olarak alan ve ilgili textBox ve label nesnesini oluşturacak olan
        KontrolOlustur isimli metodu çağırıyor.*/
        for(int i=1;i<dt.Columns.Count;++i)
        {
            KontrolOlustur(dt.Columns[i].ColumnName.ToString(),i);
        }
        dgKayitlar.DataSource=dt; /* Burada dataGrid nesnemizin DataSource özelliğini DataTable nesnemiz ile ilişkilendirerek
        dataGrid'i oluşturduğumuz tabloya bağlamış oluyoruz. */
    }
    private void btnKayitEkle_Click(object sender, System.EventArgs e)
    {
        /* Bu butona tıklandığında kullanıcının oluşturmuş olduğu kontrollere girdiği değerler, tablonun ilgili alanlarına ekleniyor ve
        sonuçlar DataGrid nesnemizde gösteriliyor. */
        string kontrol;
        /* Öncelikle yeni bir dataRow nesnesi tanımlıyoruz ve DataTable sınıfına ait NewRow metodunu kullanarak dataTable'ımızın
        bellekte işaret ettiği tabloda, veriler ile doldurulmak üzere boş bir satır açıyoruz. */
        DataRow dr;
        dr=dt.NewRow();
        /* Aşağıdaki döngü gözünüze korkutucu gelsin. Yaptığımız işlem DataTable'daki alan sayısı kadar sürecektir bir döngü. Her bir
        alana, kullanıcının ilgili textBox'ta girdiği değeri eklemek için, dr[i] satırını kullanıyoruz. dr[i] dataRow nesnesinin temsil ettiği i
        indexli alana işaret ediyor. Peki ya i indexli bu DataRow alanının dataTable'daki hangi alana işaret ettiği nereden belli. İşte
        dt.NewRow() dediğimizde, dataTable'daki alanlardan oluşan bir DataRow yani satır oluşturmuş oluyoruz. Daha sonra yapmamız
        gereken ise textBox kontrolündeki veriyi almak ve bunu DataRow nesnesindeki ilgili alana aktarmak. Bunun için formdaki tüm
        kontroller taranıyor ve her bir kontrol acaba alanlar için oluşturulmuş TextBox nesnesi mi? ona bakılıyor. Eğer öyleyse DataRow
        nesnesinin i indexli alanına bu kontrolün içerdiği text aktarılıyor.*/
        for(int i=1;i<dt.Columns.Count;++i)
        {
            kontrol="txt"+dt.Columns[i].ColumnName.ToString();
            for(int j=0;j<this.Controls.Count;++j)
            {
                if(this.Controls[j].Name==kontrol)
                {

```

```

        dr[i]=this.Controls[j].Text;
    }
}
}

```

/* Artık elimizde içindeki alanları bizim girdiğimiz veriler ile dolu bir DataRow nesne örneği var. Tek yapmamız gereken bunu dataTable nesnemizin Rows koleksiyonuna eklemek. Daha sonra ise dataGrid nesnemizi tazeleyerek görüntünün yenilenmesini ve girdiğimiz satırın burada görünmesini sağlıyoruz. */

```

        dt.Rows.Add(dr);
        .Refresh();
    }

```

Şimdi programımızı deneyelim. Ben örnek olarak Ad ,Soyad ve Birim alanlarından oluşan bir tablo oluşturdum ve iki kayıt girdi. İşte sonuç,

Tablo Oluşturmak

Tablo Adı:

Alan:

Personel

ID Primary Key
 Alan Adı: Ad Veri Tipi: System.String
 Alan Adı: Soyad Veri Tipi: System.String
 Alan Adı: Birim Veri Tipi: System.String

ID	Ad	Soyad	Birim
1	Burak Selim	ŞENYURT	Bilgi İşl
2	X Birisi	Y Soyadi	Bilgi İşl

Şekil 1. Programın çalışmasının sonucu.

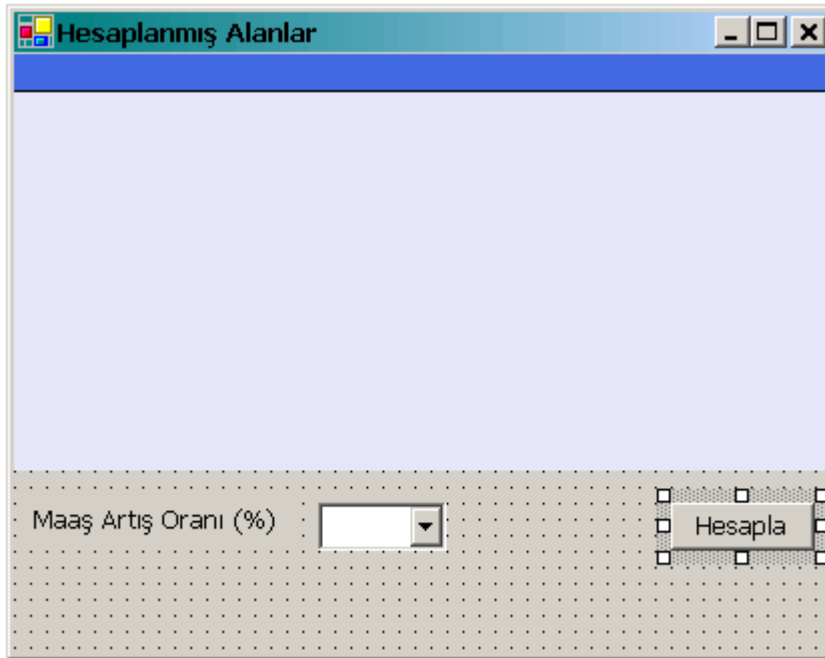
Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde, DataRelation nesnesi yardımı ile birbirleri ilişkili tabloların nasıl oluşturulacağını göreceğiz. Hepinize mutlu günler dilerim.

DataColumn.Expression Özelliği İle Hesaplanmış Alanların Oluşturulması

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde bir tabloda sonradan hesaplanan alanların nasıl oluşturulacağını incelemeye çalışacağız. Her zaman olduğu gibi konuyu iyi anlayabilmek için bir örnek üzerinden gideceğiz. Bir hesaplanan alan aslında bir hesaplama ifadesidir. Örneğin var olan bir tablodaki bir alana ait her değere ortak bir işlem yaptırmak istediğimizi ve bu her veri için oluşan sonuçlarında tabloda ayrı bir alan adı altında gözükmelerini istediğimizi varsayalım. Buna en güzel örneklerden birisi;

Diyelimki personelinizin maaş bilgilerinin tutulduğu tablomuz var. Bu tablomuzda yer alan maaş verilerinde değişik oranlarda artışlar uygulamak istediğinizi varsayalım. Yüzde 10, yüzde 15 vb...Bu artışlarda sadece ekranda izlemek istediğinizi tablo üzerinde kalıcı olarak yer almasını istemediğinizi düşünün. Bu minik problemin çözümü DataColumn sınıfına ait Expression özelliğidir. Bu özelliği yapmak istediğimiz işlemin sonucunu oluşturacak ifadelerden oluştururuz. Konuyu daha net anlayabilmek için hiç vakit kaybetmeden örneğimizde geçelim. Bu örnekte Maas isimli bir tablodaki Maas alanlarına ait değerlere kullanıcının seçtiği oranlarda artış uygulayacağız. Form tasarımı aşağıdakine benzer olacak. Burada comboBox kontrolümüzde %5 ten %55 'e kadar değerler girili. Hesapla başlıklı butona basıldığında, hesaplanan alana ilişkin işlemlerimizi gerçekleştirecek.



Şekil 1. Form Tasarımı

Hiç vakit kaybetmeden kodlarımıza geçelim.

```
SqlConnection conFriends;
```

```
SqlDataAdapter da;
```

```
DataTable dtMaas;
```

```
/* Aşağıdaki procedure ile, Sql sunucumuzda yer alan Friends isimli veritabanına bağlanıyor, buradan Maas isimli tablodaki verileri  
DataTable nesnemize yüklüyoruz. Daha sonrada dataGrid kontrolümüze veri kaynağı olarak bur DataTable nesnesimizi göstererek
```

tablonun içeriğinin görünmesini sağlıyoruz.*/

```
public void doldur()
```

```
{
```

```
    conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
```

```
    da=new SqlDataAdapter("Select * From Maas",conFriends);
```

```
    dtMaas=new DataTable("Maaslar");
```

```
    da.Fill(dtMaas);
```

```
    dgMaas.DataSource=dtMaas;
```

```
}
```

```
private void Form1_Load(object sender, System.EventArgs e)
```

```
{
```

```
    doldur(); /* Tablodan verilerimizi alan procedure'u çağırıyoruz*/
```

```
}
```

```
private void btnHesapla_Click(object sender, System.EventArgs e)
```

```
{
```

```
    /* Hesaplanan Alanımız için bir DataColumn nesnesi tanımlıyoruz. */
```

```
    DataColumn dcArtisAlani=new DataColumn();
```

/* Expression özelliğine yapmak istediğimiz hesaplamayı giriyoruz. Buradaki hesaplamada, kullanıcının cmbAris isimli comboBox kontrolünden seçtiği oran kadar maaşlara artış uygulanıyor.*/

```
    dcArtisAlani.Expression="Maas + (Maas *"+cmbArtis.Text+"/100)";
```

```
    /* Yeni alanımız için anlamlı bir isim veriyoruz. Bu isimde artış oranıda yazmakta.*/
```

```
    dcArtisAlani.ColumnName="Yuzde"+cmbArtis.Text+"Artis";
```

/* Daha sonra oluşturduğumuz bu hesaplanmış alanı DataTable nesnemize ekliyoruz. Böylece bellekteki Maas tablomuzda, maaşlara belirli bir artış oranı uygulanmış verileri içeren DataColumn nesnemiz hazırlanmış oluyor.*/

```
    dtMaas.Columns.Add(dcArtisAlani);
```

```
    /* Son olarak dataGrid nesnemizi Refresh() metodu ile tazeleyerek hesaplanmış alanımızda görünmesini sağlıyoruz.*/
```

```
    dgMaas.Refresh();
```

```
}
```

Uygulamamızı çalıştırdığımızda aşağıdaki ekran görüntüsü ile karşılaşırız. Ben örnek olarak %10 luk artış uyguladım.

PersonelID	Birim	Maas
10000	ITD	1000000000,0
10001	MU	1500000000,0
10002	IT	1250000000,0
10003	IT	1850000000,0
10004	LVZ	2250000000,0
10005	LVZ	2400000000,0
10006	GNM	3500000000,0
10007	GNM	4500000000,0

Maaş Artış Oranı (%) 10 Hesapla

Şekil 2. Program İlk Çalıştığında.

İşte işlemimizin sonuçları.

PersonelID	Birim	Maas	Yuzde10Artis
10000	ITD	1000000000,0000	1100000000,0000
10001	MU	1500000000,0000	1650000000,0000
10002	IT	1250000000,0000	1375000000,0000
10003	IT	1850000000,0000	2035000000,0000
10004	LVZ	2250000000,0000	2475000000,0000
10005	LVZ	2400000000,0000	2640000000,0000
10006	GNM	3500000000,0000	3850000000,0000
10007	GNM	4500000000,0000	4950000000,0000

Maaş Artış Oranı (%) 10 Hesapla

Şekil 3. Hesaplanmış Alan

Görüldüğü gibi %10 luk artışın uygulandığı yeni alanımız dataGrid'imiz içinde görülmektedir. Unutmayalımki bu oluşturduğumuz DataColumn nesnesi sadece bellekteki tablomuza eklenmiş bir görüntüdür. Veritabanımızdaki tablomuza doğrudan bir etisi yoktur. Tabiki performans açısından çok yüksek kapasiteli tablolarda çalışırken böyle bir işlemin yapılması özellikle ağ ortamlarında performans kaybına yol açabilir. Bunun önüne geçmek için kullanabileceğimiz yöntemlerden birisi, sql sunucusunda bu

hesaplamaların yaptırılması ve sonuçların view nesneleri olarak alınmasıdır. Ancak küçük boyutlu veya süzölmüş veriler üzerinde, Expression özelliğinin kullanımı sizinde gördüğünüz gibi son derece kolay ve faydalıdır. Geldik bir makalemizin sonuna daha, tekrar görüşünceye kadar hepinize mutlu günler dilerim.

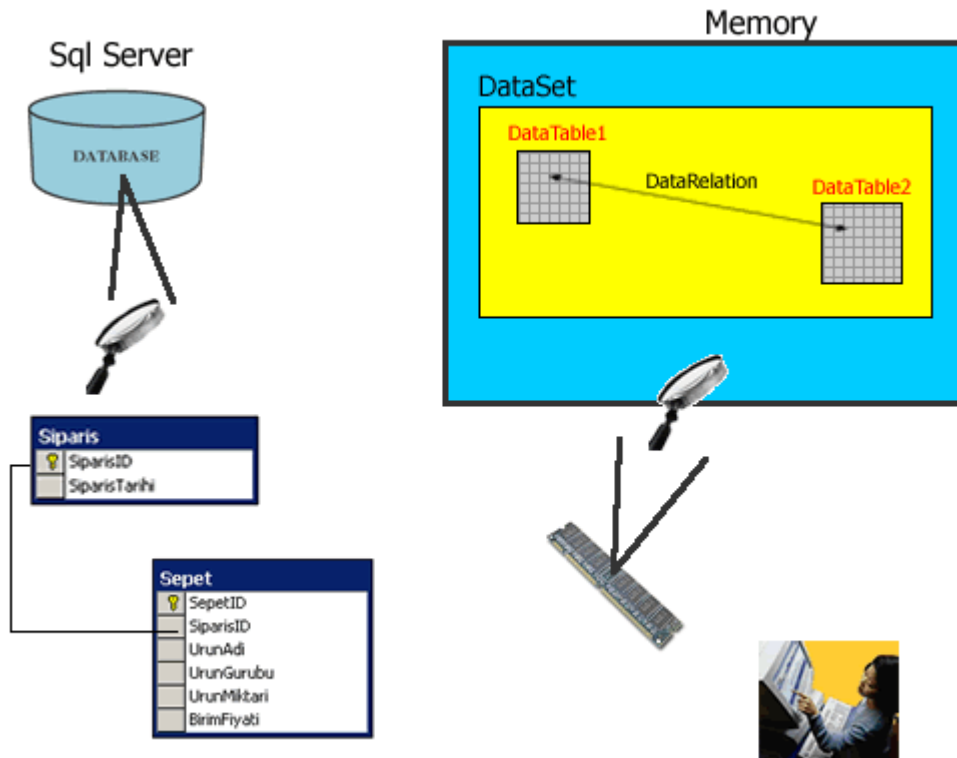
İlişkili Tabloları DataSet İle Kullanmak - 1

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, aralarında relationship (ilişki) bulunan tabloların, bir DataSet nesnesinin bellekte temsil ettiği alanda nasıl saklandığını incelemeye çalışacağız. Bunu yaparken de, geliştireceğimiz uygulama ile parant-child (ebeveyn-çocuk) yada master-detail (efendi-detay) adı verilen ilişkileri taşıyan tablolarımızı bir windows application'da bir dataGrid nesnesi ile nasıl kolayca göstereceğimizi göreceğiz.

İşin sırrı Olin'de diye bir reklam vardı eskiden. Şimdi aklıma o reklam geldi. Burada da işin sırrı DataRelation adı verilen sınıftadır. DataRelation sınıfına ait nesneler, aralarında ilişkisel bağ olan tablolarının, aralarındaki ilişkiyi temsil ederler. Bir DataRelation nesnesi kullandığımızda, bu nesneyi mutlaka bir DataSet sınıfı nesnesine eklememiz gerekmektedir. Dolayısıyla DataSet sınıfımız, aralarında ilişki olan tabloları temsil eden DataTable nesnelerini ve bu tablolar arasındaki ilişkiyi temsil eden DataRelation nesnesini(lerini) taşımak durumundadır.

Aşağıdaki şekil ile , bu konuyu zihninizde daha kolay canlandırabiliriz. Söz konusu tablolar, yazacağımız uygulamayada da kullanacağımız tablolardır. Dikkat edilecek olursa buradaki iki tablo arasında Siparis isimli tablodan, Sepet isimli tabloya bire-çok (one to many) bir ilişki söz konusudur. DataRelation nesnemiz bu ilişkiyi DataSet içinde temsil etmektedir.



Şekil 1 . DataRelation

Bir DataRelation nesnesi oluşturmak için kullanabileceğimiz Constructor metodlar şunlardır.

1 - public DataRelation(string, DataColumn, DataColumn);

2 - public DataRelation(string, DataColumn[], DataColumn[]);

3 - public DataRelation(string, DataColumn, DataColumn, bool);

4 -public DataRelation(string, DataColumn[], DataColumn[], bool);

Tüm yapıcı metodlar ilk parametre olarak DataRelation için string türde bir isim alırlar. İl yapıcı metodumuz, iki adet DataColumn tipinde parametre almaktadır. İlk parametre master tabloya atı primary key alanını, ikinci DataColumn parametresi ise detail tabloya ait secondary key alanını temsil etmektedir. İkinci yapıcı metodu ise aralarındaki ilişkiler birden fazla tablo alanına bağlı olan tablo ilişkilerini tanımlamak içindir. Dikkat edilecek olursa, DataColumn[] dizileri söz konusudur.Üçüncü ve dördüncü yapıcılarında kullanım tarzları bir ve ikinci yapıcılar ile benzer olmasına karşın aldıkları bool tipinde dördüncü bir parametre daha vardır. Dördüncü parametre , tablolar arası kullanılacak veri bütünlüğü kuralları uygulanacak ise True değerini alır eğer bu kurallar uygulanmayacak ise false değeri verilir.

Şimdi gelin kısa bir uygulama ile bu konuyu işleyelim. Uygulamamızda kullanılan tablolara ait alanlar ve özellikleri şöyledir. İlk tablomuz Siparis isimli tablomuz. Bu tabloda kullanıcının vermiş olduğu siparişin numarası ve tarihi ile ilgili bilgiler tutuluyor. Bu tablo bizim parent(master) tablomuzdur.

Columns:						
Key	ID	Name	Data Type	Size(...)	Nulls	Default
		SiparisID	int	4	☐	
		SiparisTarihi	datetime	8	☐	

Şekil 2. Siparis Tablosu

Diğer tablomuzda ise, verilen siparişin hangi ürünlerden oluştuğuna dair bilgiler yer almakta.Bu tablomuz ise bizim child tablomuzdur.

Columns:						
Key	ID	Name	Data Type	Size(...)	Nulls	Default
	☒	SepetID	int	4	☐	
		SiparisID	int	4	☐	
		UrunAdi	char	50	☐	
		UrunGurubu	char	50	☐	
		UrunMiktari	int	4	☐	
		BirimFiyati	money	8	☐	

Şekil 3. Sepet Tablosu

Uygulamamızı bir windows application olarak geliştireceğim. Bu nedenle vs.net ortamında, yeni bir windows application oluşturuyoruz. Sayfanın tasarımı son derece basit. Bir adet dataGrid nesnemiz var ve bu nesnemiz ilişkil tabloların kayıtlarını gösterecek. Dilerseniz kodlarımızı yazmaya başlayalım.

```
private void Form1_Load(object sender, System.EventArgs e)
{
    /* Önce sql sunucumuzda yer alan Friends isimli veritabanımız için bir bağlantı nesnesi oluşturuyoruz. */
    SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
    /* SqlDataAdapter nesneleri yardımıyla, Friends veritabanında yer alan Siparis ve Sepet tablolarındaki verileri alıyoruz ve
    sonrada bunları DataTable nesnelerimize aktarıyoruz.*/
    SqlDataAdapter daSiparis=new SqlDataAdapter("Select * From Siparis",conFriends);
    SqlDataAdapter daSepet=new SqlDataAdapter("Select * From Sepet",conFriends);
    DataTable dtSiparis=new DataTable("Siparisler");
    DataTable dtSepet=new DataTable("SiparisDetaylari");
    daSiparis.Fill(dtSiparis);
    daSepet.Fill(dtSepet);
    /* Şimdi ise bu iki tablo arasındaki bire çok ilişkiyi temsil edecek DataRelation nesnemizi oluşturuyoruz. */
    DataRelation drSiparisToSepet=new
    DataRelation("Siparis_To_Sepet",dtSiparis.Columns["SiparisID"],dtSepet.Columns["SiparisID"]);
    /* Artık oluşturduğumuz bu DataTable nesnelerini ve DataRelation nesnemizi DataSet nesnemize ekleyebiliriz. Dikkat edecek
    olursanız, DataRelation nesnemizi dataSet nesnemizin Relations koleksiyonuna ekledik. DataRelation nesneleri DataTable nesneleri
    gibi DataSet'e ait ilgili koleksiyonlarda tutulmaktadırlar. Dolayısıyla bir DataSet'e birden fazla tabloyu nasıl ekleyebiliyorsak birden
    fazla ilişkiyi de ekleyebiliriz. */
    DataSet ds=new DataSet();
    ds.Tables.Add(dtSiparis);
    ds.Tables.Add(dtSepet);
    ds.Relations.Add(drSiparisToSepet);
    /* Şimdi ise dataGrid nesnemizi dataSet nesnemiz ile ilişkilendirelim */
    dataGrid1.DataSource=ds.Tables["Siparisler"];
```

}

Uygulamamızı çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz. Görüldüğü gibi Siparis tablosundaki veriler görünmektedir. Lütfen satırların yanlarındaki + işaretine dikkat edelim.(Şekil 4) Bu artı işaretine tıkladığımızda oluşturmuş olduğumuz DataRelation'ın adını görürüz.(Şekil 5)

SiparisID	SiparisTarihi
10000	01.01.2003
10001	01.02.2003
10002	01.03.2003
10003	01.04.2003

Şekil 4.

SiparisID	SiparisTarihi
10000	01.01.2003
10001	01.02.2003
10002	01.03.2003
10003	01.04.2003

Şekil 5.

Bu isimler birer link içermektedir. Bu linklerden birine tıkladığımızda bu satıra ait detaylı bilgiler child tablodan(Sepet) gösterilirler. (Şekil 6)

SepetID	SiparisID	UrunAdi	UrunGurubu	UrunMiktari	BirimFiyati
1	10001	ADO.NET	Bilgisayar Kita	1	20000000,000
2	10001	Audioslave	Müzik CD	1	24500000,000
3	10001	PC World	Bilgisayar Der	1	5000000,0000

Şekil 6.

Bir sonraki makalemizde tablolar arasındaki veri bütünlüğünü sağlayan Constraint kurallarının nasıl DataSet'e aktarıldığını inceleyeceğiz.

Geldik bir makalemizin daha sonuna. Hepinize mutlu günler dilerim.

İlişkili Tabloları DataSet İle Kullanmak - 2

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde ilişkili tablolar arasında kısıtlamaların (constraints) nasıl kullanıldığını işlemey çalışacağız. Hatırlayacağınız gibi bu yazı dizisinin ilk bölümünde DataRelation sınıfını kullanarak ilişkil tabloların bellekte nasıl ifade edilebileceğini görmüştük. Bir diğer önemli konu, bu ilişkili tablolar arasındaki parent-child ilişkisinin kayıt güncelleme, kayıt silme gibi durumlarda nasıl hareket edeceğini belirlemektir.

Buna verilecek en güzel örnek müşterilere ait sipariş detaylarının ve buna benzer ilişkilerin yer aldığı veritabanı tasarımlarıdır. Söz gelimi parent tablodan bir kaydın silinmesi ile bu kayda bağlı child tablodaki kayıtların da silinmesi gerekebilir yada silinmemesi istenebilir. İşte bu noktada DataSet ile belleğe yüklenen tablolar arasındaki bu zorlamaları bir şekilde tanımlamamız gerekmektedir. Bu zorlamalar Constraints olarak tanımlanır. Bizim için en önemli iki kısıtlama Foreign Key Constraints ve Unique Constraints dir. Foreign Key Constraints (Yabancı anahtar kısıtlaması), parent-child ilişkisine sahip tablolarda kayıtların güncelleme ve silme olaylarında nasıl hareket edeceğini belirleriz. Unique Constraints tanımı ile de bir alanın değerlerinin asla tekrar edilemeyeceği şartını bildirmiş oluruz. Bir yabancı anahtar kısıtlaması için ForeignKeyConstraint sınıfı kullanılır. Aynı şekilde bir teklik kısıtlaması için de UniqueConstraints sınıfı kullanılmaktadır. Her iki sınıfa ait örnek nesnelerin kullanılabilmesi için, ilgili tablonun Constraints koleksiyonuna eklenmesi gerekmektedir.

Şöyleki, diyelimki siparişlerin tutulduğu tablodaki veriler ile sipariş içinde yer alan ürünlerin tutulduğu tablolar arasında bire-çok bir ilişki var. Bu durumda, siparişlerin tutulduğu tablodan bir kayıt silindiğinde buradaki siparişi belirleyici alan (çoğunlukla ID olarak kullanırız) ile child tabloda yer alan ve yabancı anahtar ile parent tabloya bağlı olan alanları silmek isteyebiliriz. Burada zorlama unsurumuz parent tabloyu ilgilendirmektedir. Dolayısıyla, oluşturulacak ForeignKeyConstraints nesnesini parent tablonun Constraints koleksiyonuna ekleriz. Elbette, kısıtlamanın silme ve güncelleme gibi işlemlerde nasıl davranış göstermesi gerektiğini belirlemek için, ForeignKeyConstraints'e atı bir takım özelliklerinde ayarlanması gerekir. Bunlar,

DeleteRule
UpdateRule
AcceptRejectRule

özellikleridir. Bu özelliklere atayabileceğimiz değerler ise,

Rule.Cascade
Rule.None
Rule.SetDefault

Rule.SetNull

Cascade değeri verildiğinde güncelleme ve silme işlemlerinden, child tablodaki kayıtlarında etkilenmesi sağlanmış olunur. Sözgelimi parent tabloda bir satırın silinmesi ile ilişkili tablodaki ilişkili satırlarında tümü silinir. None değeri verildiğinde tahmin edeceğimiz gibi bu değişiklikler sonunda child tabloda hiç bir değişiklik olmaz. SetDefault değeri, silme veya güncelleme işlemleri sonucunda child tablodaki ilişkili satırların alanlarının değerlerini varsayılan değerlerine (çoğunlukla veritabanında belirlenen) ayarlar. SetNull verildiğinde ise bu kez, child tablodaki ilişkili satırlardaki alanların değerleri, DBNull olarak ayarlanır.

Burada AcceptRejectRule isimli bir özellikte dikkatinizi çekmiş olmalı. Bu özellik bir DataSet, DataTable veya DataRow nesnesine ait AcceptChanges (değişiklikleri onayla) veya RejectChanges (değişiklikleri iptal et) durumunda nasıl bir kısıtlama olacağını belirlemek için kullanılır. Bu özellik Cascade veya Null değerlerinden birini alır.Bir dataSet'in kısıtlamaları uygulaması için EnforceConstraints özelliğine true değeri atanması gerektiğininide söyleyelim.

Şimdi önceki makalemizde yazdığımız örnek uygulama üzerinden hareket ederek, ForeignKeyConstraint tekniğini inceleyelim. Konuyu uzatmamak amacıyla aynı örnek kodları üzerinden devam edeceğim. Uygulamamızda kullanıcı silmek istediği Siparis'in SiparisID bilgisini elle girecek ve silme işlemini başlatıcak. İşte bu noktada, DataSet'e eklemiş olduğumuz kısıtlama devreye girerek, Sepet tablosunda yer alan ilişkili satırlarında silinmesi gerçekleştirilecek. Haydi gelin kodlarımızı yazalım.

```
SqlConnection conFriends;
SqlDataAdapter daSiparis;
SqlDataAdapter daSepet;
DataTable dtSiparis;
DataTable dtSepet;
ForeignKeyConstraint fkSiparisToSepet;
DataSet ds;
private void Form1_Load(object sender, System.EventArgs e)
{
    conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
    daSiparis=new SqlDataAdapter("Select * From Siparis",conFriends);
    daSepet=new SqlDataAdapter("Select * From Sepet",conFriends);
    dtSiparis=new DataTable("Siparisler");
    dtSepet=new DataTable("SiparisDetaylari");
    daSiparis.Fill(dtSiparis);
    daSepet.Fill(dtSepet);
    dtSiparis.PrimaryKey=new DataColumn[] {dtSiparis.Columns["SiparisID"]};
    /* Sıra geldi foreignKeyConstraint tanımlamamıza. */
    fkSiparisToSepet=new ForeignKeyConstraint("fkS_S",dtSiparis.Columns["SiparisID"],dtSepet.Columns["SiparisID"]); /*
Öncelikle yeni bir ForeignKeyConstraint nesnesi tanımlıyoruz.*/
    fkSiparisToSepet.DeleteRule=Rule.Cascade; /* Delete işleminde uygulanacak kuralı belirliyoruz.*/
    fkSiparisToSepet.UpdateRule=Rule.Cascade; /* Güncelleme işleminde uygulanacak kuralı belirliyoruz.*/
```

fkSiparisToSepet.AcceptRejectRule=AcceptRejectRule.Cascade; /* AcceptChanges ve RejectChanges metodları çağırıldığında uygulanacak olan kuralları belirliyoruz.*/

```

    ds=new DataSet();
    ds.Tables.Add(dtSiparis);
    ds.Tables.Add(dtSepet);
    ds.Tables["SiparisDetaylari"].Constraints.Add(fkSiparisToSepet); /* Oluşturduğumuz kısıtlamayı ilgili tablonun Constraints koleksiyonuna ekliyoruz. */
    ds.EnforceConstraints=true; /* Dataset'in barındırdığı kısıtlamaları uygulatmasını bildiriyoruz. False değeri atarsak dataset nesnesinin içerdiği tablo(lara) ait kısıtlamalar görmezden geliriz.*/
    dataGrid1.DataSource=ds;
}
private void btnSil_Click(object sender, System.EventArgs e)
{
    try
    {
        DataRow CurrentRow=dtSiparis.Rows.Find(txtSiparisID.Text);
        CurrentRow.Delete();
    }
    catch(Exception hata)
    {
        MessageBox.Show(hata.Source+": "+hata.Message);
    }
}

```

Şimdi uygulamamızı çalıştıralım. Siparis tablosuna baktığımızda aşağıdaki görünüm yer almaktadır.

SiparisID	SiparisTarihi
10000	01.01.2003
10001	01.02.2003
10002	01.03.2003
10003	01.04.2003

SiparisID

Şekil 1. Sipariş Verileri

Siparis Detaylarının tutulduğu Sepet tablosunda ise görünüm şöyledir.

SepetID	SiparisID	UrunAdi	UrunGurubu	UrunMiktari	BirimFiyati
1	10001	ADO.NET	Bilgisayar Kita	1	20000000,000
2	10001	Audioslave	Müzik CD	1	24500000,000
3	10001	PC World	Bilgisayar Der	1	5000000,0000
4	10002	Lacoste	Triko Kazak	2	18000000,000
5	10002	Levis	Kot Pantolon	1	7800000,0000
6	10003	Windows XP	Isletim Sistem	2	25000000,000
7	10000	Nokia 3110	Cep Telefonu	4	100000000,00
8	10000	Siemens Giga	Telsiz Telefon	1	17800000,000

SiparisID

Şekil 2. Sipariş Detayları

Şimdi 10002 nolu sipariş satırını silelim. Görüldüğü gibi Sepet tablosundan 10002 ile ilgili tüm ilişkil kayıtlarda silinecektir. Aynı zamanda Siparis tablosundan da bu siparis numarasına ait satır silinecektir. Elbette dataSet nesnemize ait Update metodunu kullanmadığımız için bu değişiklikler sql sunucumuzdaki orjinal tablolara yansımayacaktır.

NewDataSet						
	SepetID	SiparisID	UrunAdi	UrunGurubu	UrunMiktari	BirimFiyati
▶	1	10001	ADO.NET	Bilgisayar Kita	1	20000000,000
	2	10001	Audioslave	Müzik CD	1	24500000,000
	3	10001	PC World	Bilgisayar Der	1	50000000,0000
	6	10003	Windows XP	İsletim Sistem	2	250000000,000
	7	10000	Nokia 3110	Cep Telefonu	4	100000000,00
	8	10000	Siemens Giga	Telsiz Telefon	1	178000000,000
*						

SiparisID

Şekil 3. 10002 nolu siparise ait tüm kayıtlar, Sepet tablosundan Silindi.

Form1		
NewDataSet		
	SiparisID	SiparisTarihi
▶	10000	01.01.2003
	10001	01.02.2003
	10003	01.04.2003
*		

Şekil 4. 10002 Siparis tablosundan silindi.

Şimdi oluşturmuş olduğumuz bu kısıtlamada Delete kuralını None yapalım ve bakalım bu kez neler olucak. Bu durumda aşağıdaki hata mesajını alacağız.

System.Data:Cannot delete this row because constraints are enforced on relation fkS_S, and deleting this row will strand child rows.

OK

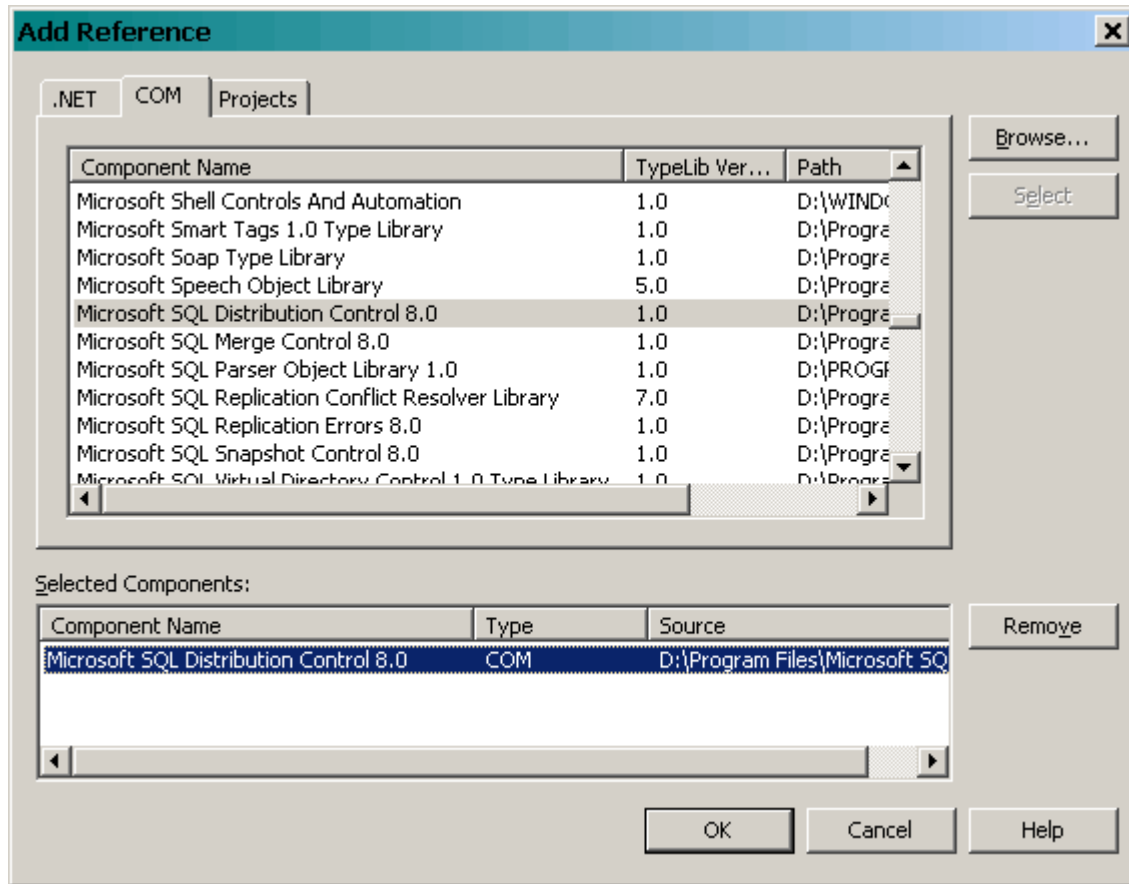
Şekil 5. DeleteRule=Rule.None

Doğal olaraktanda, ne Siparis tablosundan ne de Sepet tablosundan satır silinmeyecektir. Bu kısa bilgilerden sonra umuyorumki kısıtlamalar ile ilgili kavramlarkafanızda daha net bir şekilde canlanmaya başlamıştır. Geldik bir makalemizin daha sonuna bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

SQL_DMO İşlemleri

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, Sql Distributed Management Objects (SQL Dağıtık Yönetim Nesneleri) kütüphanesini incelemeye çalışacağız. SQL_DMO kütüphanesi bir COM uygulaması olup, SQL sunucusu ile birlikte sisteme kurulmaktadır. Bu kütüphanedeki sınıflar yardımıyla, var olan bir sql sunucusu üzerinde yönetimsel işlemler gerçekleştirebiliriz. Örneğin, kullanıcı tanımlayabilir, yeni bir veritabanı yaratabilir bu veritabanına ait tablolar oluşturabilir, var olan bir veritabanı için yedekleme işlemleri gerçekleştirebilir, yedeklenmiş veritabanılarını geri yükleyebilir ve bunlar gibi pek çok yönetsel işlemi gerçekleştirebiliriz. Uygulamalarımızda bu tip işlemleri kodlayabilmek için, Microsoft SQL Distributin Control'un projemize referans edilmesi gerekmektedir. Bir uygulamaya bunu referans etmek için VS.NET ortamında, Add Reference kısmında, COM nesneleri altında Microsoft SQL Distribution Control 8.0 seçilmelidir. Aşağıdaki şekilde bunun nasıl yapıldığını görebilirsiniz.



Şekil 1. Microsoft SQL Distribution Control 8.0 'in eklenişi.

Bu noktadan sonra uygulamamızda SQLDMO isimli temel sınıfı kullanarak bahsetmiş olduğumuz işlemleri gerçekleştirebiliriz. Konuyu daha iyi kavrayabilmek amacıyla derseniz, hemen basit bir uygulama gerçekleştirelim. Bu uygulamamızda, Sql sunucumuz üzerinde, bir veritabanı yaratacak bu veritabanı içinde çok basit bir tablo oluşturacak, Sql sunucusunda yer alan veritabanlarını göreceğiz ve bu veritabanlarına ait tablolara bakabileceğiz. Kodların işleyişini incelediğinizde, işin püf noktasının SQLDMO sınıfında yer alan SQLServerClass, DatabaseClass, TableClass, ColumnClass sınıflarında olduğunu görebilirsiniz. Buradaki matnık aslında

ADO.NET mantığı ile tamamen aynıdır. SQL Sunucusuna bağlanmak için kullanacağınız bir SQLServerClass sınıfı, bir veritabanını temsil eden DatabaseClass sınıfı, bir tablo için TableClass sınıfı ve tabloya ait alanları temsil edecek olan ColumnClass sınıfı vardır.

Matnık aynı demiştik. Bir veritabanı yaratmak için, DatabaseClass sınıfından örnek bir nesne oluşturursunuz. Bunu var olan Sql sunucusuna eklemek demek aslında bu nesneyi SQLServerClass nesnenizin Databases koleksiyonuna eklemek demektir. Aynı şekilde bir tablo oluşturmak için TableClass sınıfı örneğini kullanır,ve bunu bu kez DatabaseClass nesnesinin Tables koleksiyonuna eklersini. Tahmin edeceğimiz gibi bir tabloya bir alan eklemek için ColumnClass sınıfından örnek bir nesne kullanır ve bunun özelliklerini ayarladıktan sonra tablonun Columns koleksiyonuna eklersiniz. Kodlarımızı incelediğiniz zaman konuyu çok daha net bir şekilde anlayacaksınız. Uygulamamız aşağıdakine benzer bir formdan oluşmakta. Sizde buna benzer bir form oluşturarak işe başlayabilirsiniz.

Şekil 2. Form tasarımıımız.

Şimdi kodlarımızı yazalım.

```
SQLDMO.SQLServerClass srv;
SQLDMO.DatabaseClass db;
SQLDMO.TableClass tbl;
private void btnConnect_Click(object sender, System.EventArgs e)
{
    srv=new SQLDMO.SQLServerClass(); /* SQL Sunucusu üzerinde, veritabani yaratma gibi işlemler için, Sql Sunucusunu temsil edecek ve ona bağlanmamızı sağlayacak bir nesneye ihtiyacımız vardır. Bu nesne SQLDMO sinifinda yer alan SQLServerClass sinifinin bir örneği olacaktır. */
```

srv.LoginSecure=false; /* Bu özellik true olarak belirlendiğinde, Sql Sunucusuna Windows Authentication seklinde baglanilir. Eger false degerini verirsek bu durumda Sql Server Authentication geçerli olur. Iste bu durumda SQLServerClass nesnesini Connect metodu ile Sql Sunucusuna baglanirken geçerli bir kullanıcı adı ve şifre girmemiz gerekmektedir. */

```
try
{
    srv.Connect("BURKI","sa","CucP??80."); /* Baglanti kurmak için kullandigimiz Connect metodu üç parametre almaktadır. İlk parametre sql sunucusunun adidir. İkinci parametre kullanıcı adı ve üçüncü parametrede şifresidir. Eger LoginSecure=true olarak ayarlasaydik, kullanıcı adini ve şifreyi bos birakicaktik, nitekim Windows Authentication (windows dogrulaması) söz konusu olucakti.*/
    durumCubugu.Text="Sunucuya baglanildi..." +srv.Status.ToString();
}
catch(Exception hata)
{
    MessageBox.Show(hata.Message);
}
}

private void btnVeritabaniOlustur_Click(object sender, System.EventArgs e)
{
    try
    {
        db=new SQLDMO.DatabaseClass(); /* SQL-DMO kütüphanesinde, veritabanlarini temsil eden sinif DatabaseClass sinifidir. */
        db.Name=this.txtVeritabaniAdi.Text; /* Veritabani nesnemizin name özelligi ile veritabaninin adı belirlenir.*/
        srv.Databases.Add(db); /* olusturulan DatabaseClass nesnesi SQLServerClass sinifinin Databases koleksiyonuna eklenerek Sql Sunucusu üzerinde olusturulmasi saglaniyor. */
        durumCubugu.Text=db.Name.ToString()+" veritabani " +srv.Name.ToString()+" SQL Sunucusunda olusturuldu";
    }
    catch(Exception hata)
    {
        MessageBox.Show(hata.Message);
    }
}

private void btnTabloOlustur_Click(object sender, System.EventArgs e)
{
    try
    {
        tbl=new SQLDMO.TableClass(); /* Yeni bir tablo olusturabilmek için SQL-DMO kütüphanesinde yer alan, TableClass sinifi kullanilir.*/
        tbl.Name=txtTabloAdi.Text; /* Tablomuzun ismini name özelligi ile belirliyoruz.*/
        SQLDMO.ColumnClass dc; /* Tabloya eklenecek alanların her birisi birer ColumnClass sinifi nesnesidir. */
```

```

        dc=new SQLDMO.ColumnClass();/* Bir ColumnClass nesnesi yaratiliyor ve bu nesnenin gerekli özellikleri belirleniyor. Name
özelligi ile ismi, Datatype özelliği ile veri türü belirleniyor. Biz burada ID isimli alanımızın otomatik olarak artan ve 1000 den
başlayarak 1'er artan bir alan olmasını istedik. */
        dc.Name="ID";
        dc.Datatype="Int";
        dc.Identity=true;
        dc.IdentitySeed=1000;
        dc.IdentityIncrement=1;
        tbl.Columns.Add(dc); /* Oluşturulan bu alan TableClass nesnemizin Columns koleksiyonuna eklenerek tabloda oluşturulması
sağlanmış oluyor. */
        dc=new SQLDMO.ColumnClass();
        dc.Name="ISIM";
        dc.Datatype="char"; /* String tipte bir alan */
        dc.Length=50;
        tbl.Columns.Add(dc);
        dc=new SQLDMO.ColumnClass();
        dc.Name="SOYISIM";
        dc.Datatype="char";
        dc.Length=50;
        tbl.Columns.Add(dc);
        /* Son olarak oluşturulan TableClass nesnesi veritabanımızın tables koleksiyonuna ekleniyor. Böylece Sql sunucusunda yer
alan veritabanı içinde oluşturulması sağlanmış oluyor. */
        db.Tables.Add(tbl);
        durumCubugu.Text=tbl.Name.ToString()+" oluşturuldu...";
    }
    catch(Exception hata)
    {
        MessageBox.Show(hata.Message);
    }
}

private void btnSunucuVeritabanlari_Click(object sender, System.EventArgs e)
{
    this.lstDatabases.Items.Clear();
    /* Öncelikle listBox nesnemize Sql Sunucusunda yer alan veritabanlarının sayısını aktarıyoruz.*/
    this.lstDatabases.Items.Add("Sunucudaki veritabanı sayısı="+srv.Databases.Count);
    /* Şimdi bir for döngüsü ile, srv isimli SQLServerClass nesnemizin Databases koleksiyonunda geziniyoruz ve her bir
databaseClass nesnesinin adını alıp listBox nesnemize aktarıyoruz. Burada index değerinin 1 den başladığına sıfırdan başlamadığına
dikkat edelim. */
    for(int i=1;i<srv.Databases.Count;++i)

```

```

{
    this.lstDatabases.Items.Add(srv.Databases.Item(i,srv).Name.ToString());
}
}
private void btnTablolar_Click(object sender, System.EventArgs e)
{
    /* Burada seçilen veritabanına ait tablolar listBox kontrolüne getiriliyor */
    this.lstTables.Items.Clear();
    this.lstTables.Items.Add("Tablo Sayisi="+srv.Databases.Item(this.lstDatabases.SelectedIndex,srv).Tables.Count.ToString());
    /* Döngümüz Sql Suncusundan yer alan veritabanı sayısı kadar süren bir döngü. */
    for(int i=1;i<srv.Databases.Item(this.lstDatabases.SelectedIndex,srv).Tables.Count;++i)
    {
        this.lstTables.Items.Add(srv.Databases.Item(this.lstDatabases.SelectedIndex,srv).Tables.Item(i,srv). Name.ToString());
    }
}

```

Şimdi uygulamamızı çalıştıralım ve öncelikle Sql Sunucumuza bağlanalım.



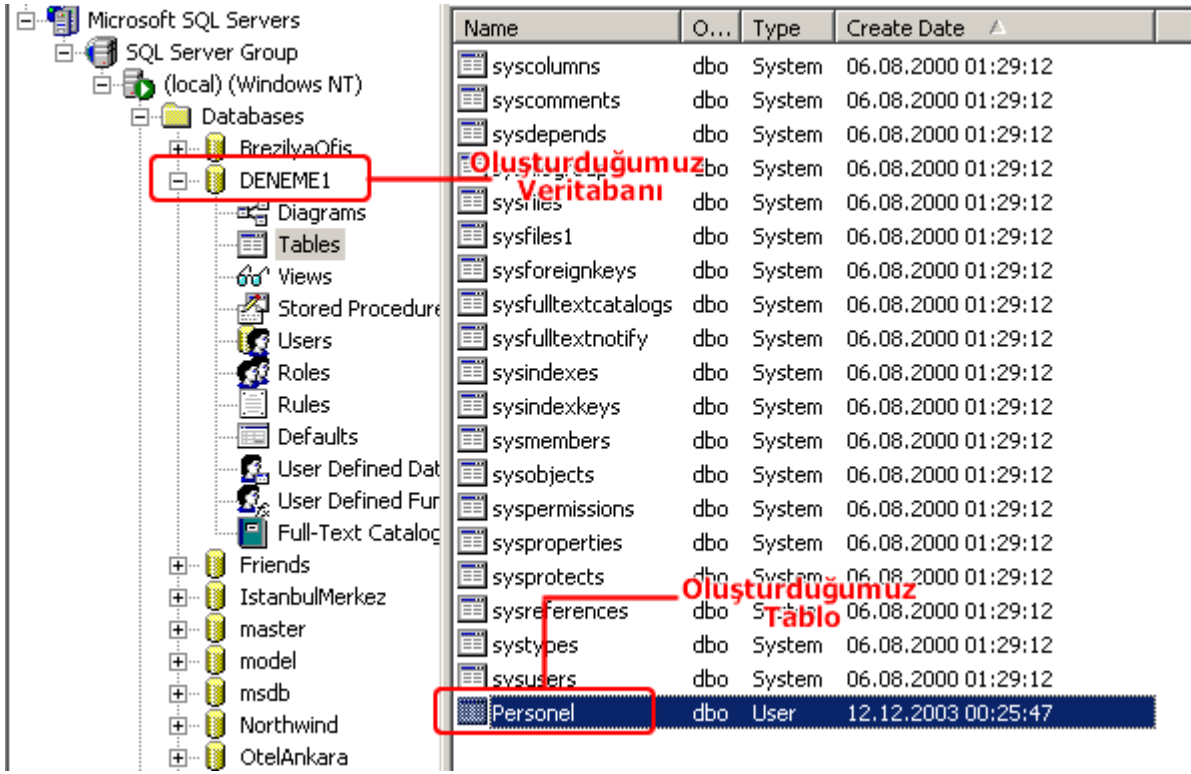
Şekil 3. Sunucumuza Bağlandık.

Şimdi bir veritabanı oluşturalım. Ben veritabanı adı olarak DENEME1 yazdım.



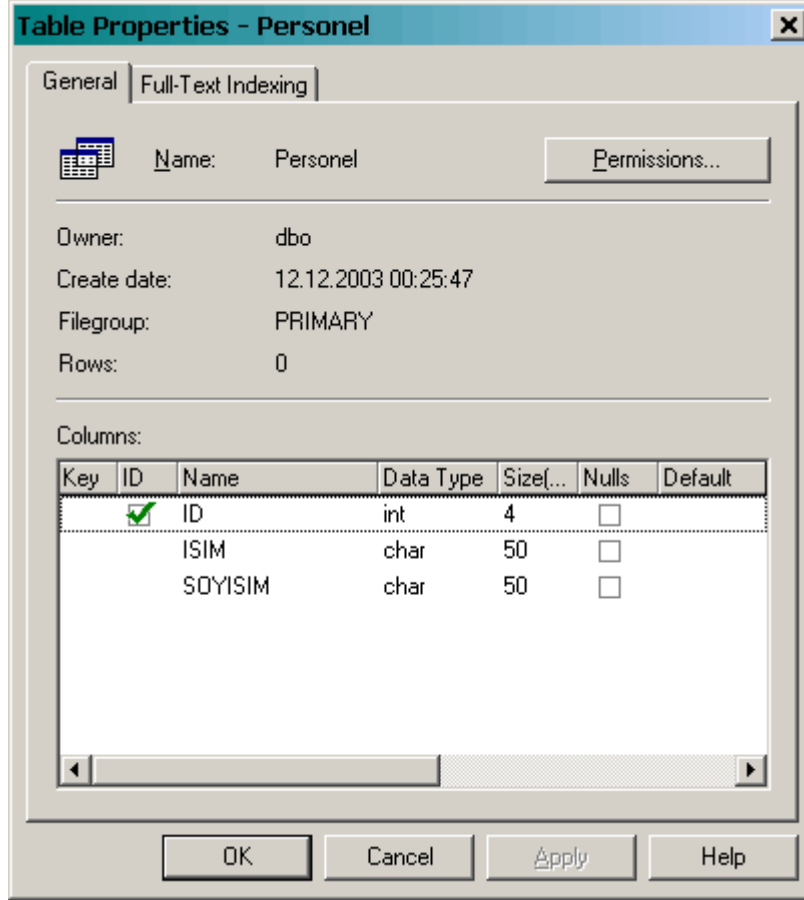
Şekil 4. Veritabanımız Oluşturuldu.

Şimdi ise tablo ismimizi yazalım. Ben deneme olarak Personel yazdım. Şimdi dilerseniz Sql Sunucumuzu bir açalım ve bakalım veritabanımız ve ilgili tablomu yaratılmışmı.



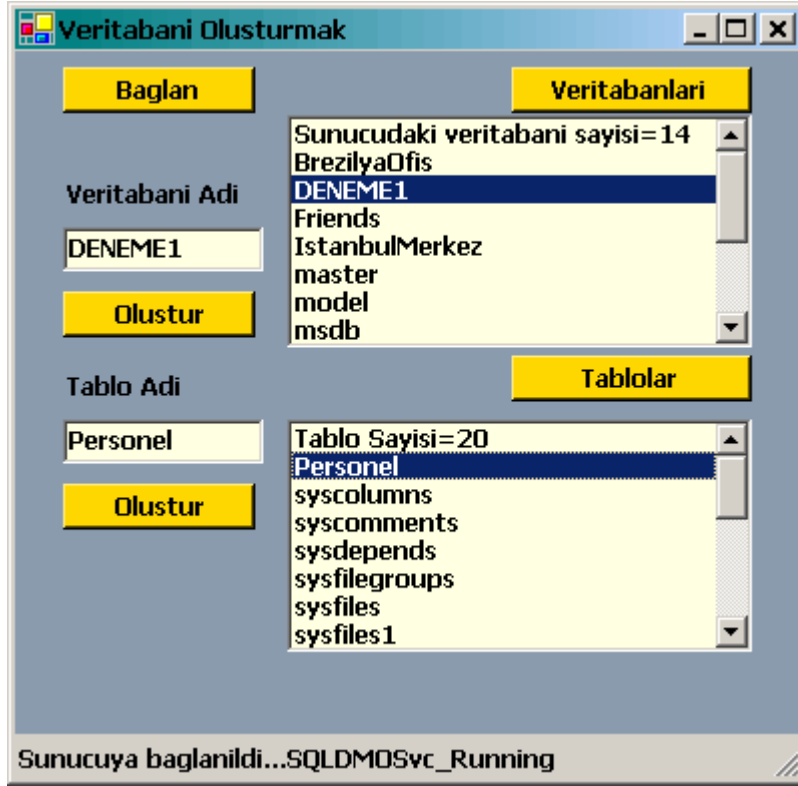
Şekil 5. Veritabanımız ve Tablomuz oluşturuldu.

Ve tablomuz baktığımızda oluşturduğumuz alanları görebiliriz.



Şekil 6. Tablomuzdaki alanlar.

Son olarak suncumuzda yer alan veritabanlarının ve Deneme veritabanı altındaki tablolarında programımızda nasıl görüldüğüne bakalım. Dikkat ederseniz tablolar ekrana geldiğinde sistem tablolarında gelir. Bununla birlikte veritabanları görünürken sadece TempDB veritabanı görünmez.



Şekil 7. Program ekranını son görüntüsü

Evet geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle, hepinize mutlu günler dilerim.

DataView Sınıfı ve Faydaları

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde getirileri ve performansı ile ADO.NET içerisinde önemli bir yere sahip olan DataView nesnesini incelemeye çalışacağız. Özellikle bu sınıfa ait, RowFilter özelliğinin ne gibi faydalar sağlayacağına da değineceğiz.

DataView sınıfı, veritabanı yönetim sistemlerindeki view nesneleri göz önüne alınarak oluşturulmuş bir sınıftır. Bilindiği gibi veritabanı yönetim sistemlerinde (DBMS-DataBase Management System) , bir veya birden fazla tablo için gerçekleştireceğimiz çeşitli tipteki birleştirici veya ayrı ayrı sorgulamalar sonucu elde edilen veri kümelerini view nesnelere aktarabilmekteyiz. View nesneleri sahip oldukları veri alt kümelerini temsil eden birer veritabanı nesnesi olarak, önceden derlendiklerinden, süratli ve performansı yüksek yapılardır. Söz gelimi her hangibi tabloya ait bir filtreleme işini bir view nesnesinde barındırabiliriz. Bunun sonucu olarak, aynı sorguyu yazıp çalıştırmak, view nesnesinin içeriğine(yani sahip olduğu verilere) bakmaktan çok daha yavaştır. Bununla birlikte bu sorgulamanın birden fazla tabloyu içerdiğini düşünürsek, bu durumda da çalıştırılan sorgu sonucu elde edilecek veri alt kümelerini bir(birkaç) view nesnesinde barındırmak bize performans, hız olarak geri dönecektir.

Gelelim ADO.NET'e. ADO.NET içerisinde de, view lara benzer bir özellik olarak DataView nesneleri yer almaktadır. DataView nesneleri, veritabanı yönetim sistemlerinde yer alan view'lar ile benzerdir. Yine performans ve hız açısından avantajlıdır. Bunların yanında bir DataView nesnesi kullanılabilmek, mutlaka bir DataTable nesnesini gerektirmektedir. Nitekim DataView nesnesinin sahip olacağı veri alt kümeleri bu dataTable nesnesinin bellekte işaret ettiği tablo verileri üzerinden alınacaktır. DataView nesnesinin kullanımının belkide en güzel yeri şudur; bir DataTable nesnesinin bellekte işaret ettiği tablodan, bir den fazla görünüm elde

ederekten, bu farklı görünümünü birden fazla kontrole bağlayarak, ekranda aynı anda tek bir tablonun verilerine ait birden fazla veri kümesini izlememiz mümkün olabilmektedir. İşte bu, bence DataView nesnesi(lerini) kullanmanın ne kadar faydalı olduğunu göstermektedir.

Bilindiği gibi DataTable nesnesine ait Select özelliğine ifadeler atayarakta var olan bir tablodan veri alt kümeleri elde edebiliyorduk. Fakat bu select özelliğine atanan ifade sonucu elde edilen veriler bir DataRow dizisine aktarılıyor ve kontrollere bağlanamıyordu. Oysaki DataView sonuçlarını istediğiniz kontrole bağlamanız mümkündür. DataTable ile DataView arasında yer alan bir farkta, DataTable'ın sahip olduğu satırlar DataRow sınıfı ile temsil edilirken DataView nesnesinin sahip olduğu satırlar DataRowView sınıfı ile temsil edilirler. Bir DataTable nesnesine nasıl ki yeni satırlar ekleyebiliyor, silebiliyor ve primary key üzerinden arama yapabiliyorsak aynı işlemleri DataView nesnesi içinde yapabiliriz. Bunları AddNew, Delete ve Find yöntemleri ile yapabiliriz. Bir sonraki makalemizde bu metodlar ile ilgili geniş bir örnek daha yapacağız.

Bugünkü makalemizde konuya açıklık getirmesi açısından iki adet örnek yapacağız. Her iki örneğimizde ağırlıklı olarak DataView nesnesinin RowFilter özelliği üzerinde duracak. RowFilter özelliği DataTable sınıfının Select özelliğine çok benzer. Bir süzme ifadesi alır. Oluşturulan ifade içinde, kullanılacak alan(alanların) veri tiplerine göre bazı semboller kullanmamız gerekmektedir. Bunu açıklayan tablo aşağıda belirtilmiştir.

Veri Tipi	Kullanılan Karakter	Örnek
Tüm Metin Değerleri	' (Tek tırnak)	" Adi='Burak' "
Tarihsel Değerler	#	" DogumTarihi=#04.12.1976# "
Sayısal Değerler	Hiçbirşey	" SatisTutari>150000000"

Tablo 1. Veritipine göre kullanılacak özel karakterler

Diğer yandan RowFilter özelliğinde IN, Like gibi işlemler kullanarak çeşitli değişik sorgular elde edebiliriz. Mantıksal birleştiriciler yardımıyla(and,or...) birleşik ifadeler oluşturabiliriz. Aslında RowFilter özelliği sql'de kullandığımız ifadeler ile aynıdır. Örnekler verelim;

Kullanılan İşleç	Örnek	Ne Yapar?
IN	" PUAN IN(10,20,25) "	PUAN isimli alan 10, 20 veya 25 olan satırlar.
LIKE	" ADI LIKE 'A*' "	ADI A ile başlayanlar (* burada asteriks karakterimizdir.)

Tablo 2. İşleçler.

Şimdi gelin konuyu daha iyi anlayabilmek amacıyla örneklerimize geçelim. Konuyu anlayabilmek için iki farklı örnek yapacağız. İlk örneğimizde, aynı DataTable için farklı görünümde elde edip bunları kontrollere bağlayacağız. İlk örneğimizde, çeşitli ifadeler kullanıp değişik alt veri kümeleri alacağız. İşte kodlarımız,

```
SqlConnection conFriends;
```

```
SqlDataAdapter da;
```

```
DataTable dtKitaplar;
```

```
/* Baglan metodumuz ile SqlConnection nesnemizi oluşturarak, sql sunucumuza ve Friends isimli veritabanımıza bağlanıyoruz.
```

```
Daha sonra ise SqlDataAdapter nesnemiz vasıtasıyla Kitaplar isimli tablodan tüm verileri alıp DataTable nesnemize yüklüyoruz. */
```

```
public void Baglan()
```

```
{
```

```
    conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
```

```
    da=new SqlDataAdapter("Select * From Kitaplar",conFriends);
```

```
    dtKitaplar=new DataTable("Kitap Listesi");
```

```
    da.Fill(dtKitaplar);
```

```
}
```

```
private void Form1_Load(object sender, System.EventArgs e)
```

```
{
```

```
    Baglan();
```

DataGridView dvTum_Kitaplar=dtKitaplar.DefaultView; /* Bir DataTable nesnesi yaratıldığı zaman, standart olarak en az bir tane görünüme sahiptir. Bu varsayılan görünüm bir DataGridView nesnesi döndüren DefaultView metodu ile elde edilebilir. Çalıştırdığımız sql sorgusu Kitaplar tablosundaki tüm kayıtları aldığından buradaki DefaultView'da aynı veri kümesini sahip olacak bir DataGridView nesnesi döndürür. Biz bu dönen veri kümesini dvTum_Kitaplar isimli DataGridView nesnesine aktardık. Daha sonra ise DataGridView nesnemizi dgTum_Kitaplar isimli dataGridView nesnemize bağladık.*/

```
    dgTum_Kitaplar.DataSource=dvTum_Kitaplar;
```

/* Yeni bir DataGridView nesnesini yapılandırıcısının değişik bir versiyonu ile oluşturuyoruz. Bu yapılandırıcı 4 adet parametre alıyor. İlk parametremiz dataTable nesnemiz, ikinci parametremiz RowFilter ifademiz ki burada Adi alanı B ile başlayanları buluyor, üçüncü parametremiz sıralamanın nasıl yapılacağı ki burada Adi alanında göre tersten sıralama yapıyor. Son parametre ise, DataGridViewRowState türünden bir parametre. Bu özellik DataGridView içerisinde ye alan her bir DataRowView'un (yani satırın) durumunun değerini belirtir. Alacağı değerler

- * 1. Added (Sadece DataGridView'a eklenen satırları ifade eder)
- * 2. Deleted (Sadece DataGridView'dan silinmiş satırları ifade eder)
- * 3. CurrentRows (O an için geçerli tüm satırları ifade eder)
- * 4. ModifiedCurrent (Değiştirilen satırların o anki değerlerini ifade eder)
- * 5. ModifiedOriginal (Değiştirilen satırların orjinal değerlerini ifade eder)
- * 6. Unchanged (Herhangibir değişikliğe uğramamış satırları ifade eder)
- * 7. OriginalRows (Tüm satırların asıl değerlerini ifade eder)
- * 8. None (Herhangibir satır döndürmez)

Buna göre bizim DataGridView nesnemiz güncel satırları döndürecektir. */

```
DataView dvBileBaslayan=new DataView(dtKitaplar,"Adi Like 'B*'", "Adi Desc",DataViewRowState.CurrentRows);
dgA.DataSource=dvBileBaslayan;
```

/* Şimdi ise 2002 yılı ve sonrası Basım tarihine sahip verilerden oluşan bir DataView nesnesi oluşturuyoruz. Bu kez yapıcı metodumuz sadece DataTable nesnemizi parametre olarak aldı. Diğer ayarlamaları RowFilter,Sort özellikleri ile yaptık. Sort özelliğimiz sıralama kriterimizi belirliyor.*/

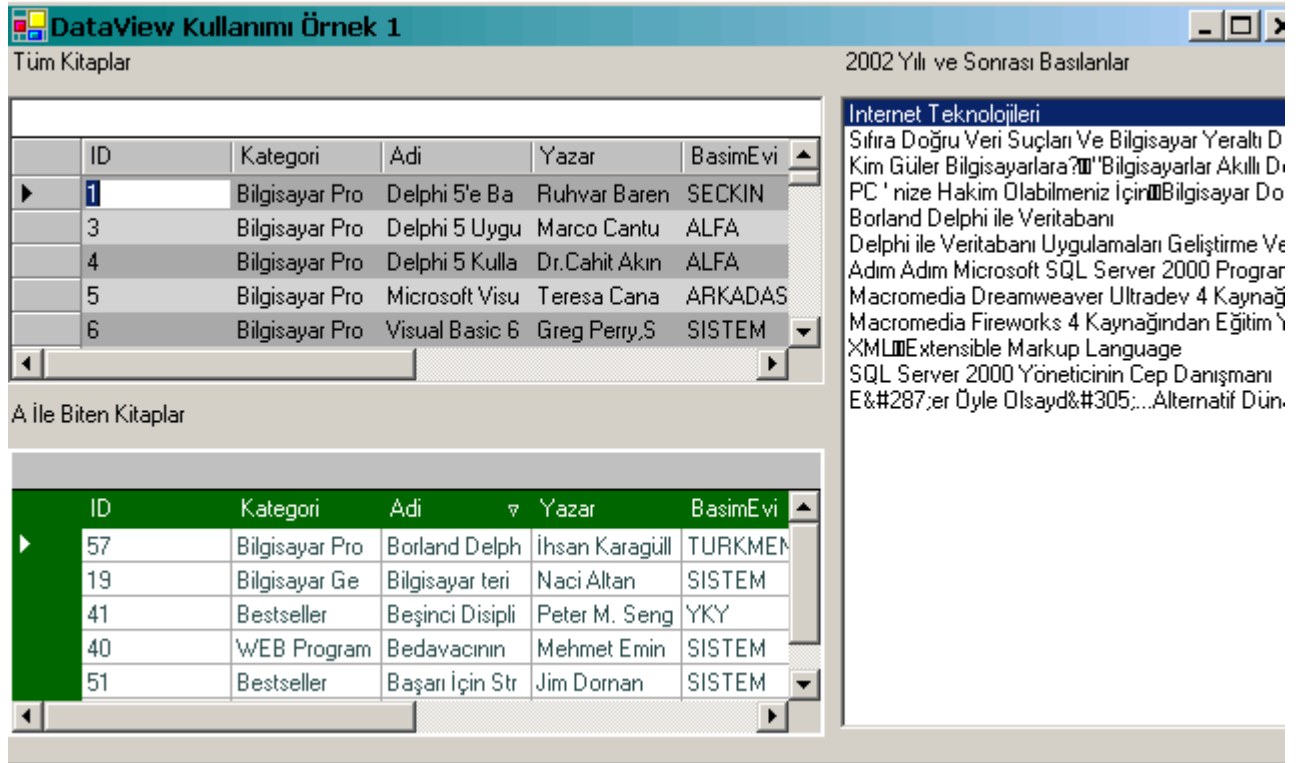
```
DataView dv2002Sonrasi=new DataView(dtKitaplar);
dv2002Sonrasi.RowFilter="BasimTarihi>=#1.1.2002#";
dv2002Sonrasi.Sort="BasimTarihi Asc";
```

/* Bu kez DataView nesnemizi bir ListBox kontrolüne bağladık ve sadece Adi alanı değerlerini göstermesi için ayarladık.*/

```
IstPahali.DataSource=dv2002Sonrasi;
IstPahali.DisplayMember="Adi";
```

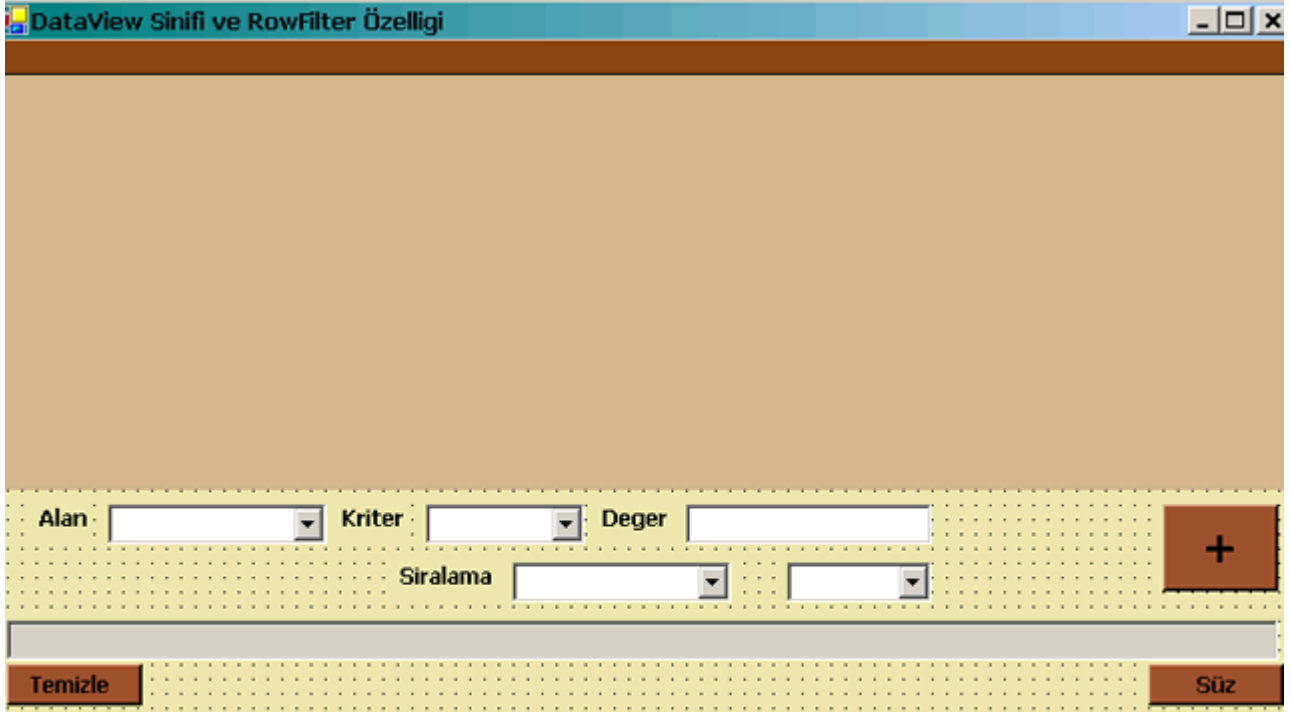
```
}
```

Çalışma sonucu ekran görüntümüz şekil 1'deki gibi olur.



Şekil 1. İlk Programın Sonucu

Şimdi gelelim ikinci uygulamamıza. Bu uygulamamızda yine Kitaplar tablosunu ele alacağız. Bu kez RowFilter özelliğine vereceğimiz ifadeyi çalışma zamanında biz oluşturacağız. Alanımızı seçecek, sıralama kriterimizi belirleyecek,aranacak değeri gireceğiz. Girdiğimiz değerlere göre program kodumuz bir RowFilter Expression oluşturacak. Programın ekran tasarımını ben aşağıdaki gibi yaptım. Sizde buna benzer bir tasarım ile işe başlayın.



Şekil2. Form Tasarımı

Şimdide kodlarımızı yazalım.

```

SqlConnection conFriends;
SqlDataAdapter da;
DataTable dtKitaplar;
DataView dvKitaplar;
/* Bu metod cmbAlanAdi isimli comboBox kontrolünü, dataTable nesneminin bellekte temsil ettiği tablonun Alanları ile doldurur.
Nitekim bu alanları, RowFilter özelliğinde kullanacağız. */
public void AlanDoldur()
{
    for(int i=0;i<dtKitaplar.Columns.Count;++i)
    {
        this.cmbAlanAdi.Items.Add(dtKitaplar.Columns[i].ColumnName.ToString());
        this.cmbAlanSira.Items.Add(dtKitaplar.Columns[i].ColumnName.ToString());
    }
}
private void Form1_Load(object sender, System.EventArgs e)
{
    conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
    da=new SqlDataAdapter("Select Kategori,Adi,Yazar,BasimEvi,BasimTarihi,Fiyat From Kitaplar",conFriends);
    dtKitaplar=new DataTable("Kitap Listesi");

```

```

/* DataTable nesnemizin bellekte temsil ettigi alanı, Kitaplar tablosundaki veriler ile, SqlDataAdapter nesnemizin Fill metodu
sayesinde dolduruyoruz.*/
da.Fill(dtKitaplar);
dvKitaplar=new DataView(dtKitaplar); /* Dataview nesnemizi yaratıyoruz. Dikkat ederseniz yapıcı metod, paremetre olarak
DataTable nesnemizi alıyor. Dolayısıyla DataView nesnemiz, dataTable içindeki veriler ile dolmuş şekilde oluşturuluyor.*/
dataGrid1.DataSource=dvKitaplar; /* DataGrid kontrolümüze veri kaynagi olarak, DataView nesnemizi isaret ederek, DataView
içindeki verileri göstermesini sagliyoruz.*/
AlanDoldur();
}

/* Bu butona bastigimizda, kullanıcının seçtiği alan, filtreleme kriteri ve filtreleme için kullanılacak deger verileri belirlenerek,
DataView nesnesinin RowFilter metodu için bir syntax belirleniyor.*/
private void btnCreateFilter_Click(object sender, System.EventArgs e)
{
    string secilenAlan=cmbAlanAdi.Text;
    string secilenKriter=cmbKriter.Text;
    string deger="";
    /* If kosullu ifadelerinde, seçilen alanin veri tipine bakıyoruz. Nitekim RowFilter metodunda, alan'in veri tipine göre ifademiz
degisiklik gösteriyor. Tarih tipindeki verilerde # karakteri aranan metnin basina ve sonuna gelirken, string tipinde degerlerde '
karakterli geliyor. Sayisal tipteki degerler için ise herhangi bir karakter ifadenin aranan degerin basina veya sonuna eklenmiyor. */
    if(dtKitaplar.Columns[secilenAlan].DataType.ToString()=="System.String")
        deger=""+txtDeger.Text+"";
    if(dtKitaplar.Columns[secilenAlan].DataType.ToString()=="System.DateTime")
        deger="#" + txtDeger.Text + "#";
    if(dtKitaplar.Columns[secilenAlan].DataType.ToString()=="System.Decimal")
        deger=txtDeger.Text;
    txtFilter.Text=secilenAlan+secilenKriter+deger; /* Oluşturulan ifade görmemiz için textBox kontrolümüze yaziliyor. */
}

private void btnFilter_Click(object sender, System.EventArgs e)
{
    dvKitaplar.RowFilter=txtFilter.Text; /* DataView nesnemizin RowFilter metoduna, ilgili ifademiz atanarak, süzme islemini
gerçekleştirmiş oluyoruz. */
    dvKitaplar.Sort=cmbAlanSira.Text+" "+cmbSiralamaKriteri.Text; /* Burada ise Sort özelliğine sıralama yapmak için gerekli
veriler ataniyor. */
}

```

Şimdi uygulamamızı çalıştıralım ve deneyelim.

Kategori	Adı	Yazar	BasımEvi	BasımTarihi	Fiyat
Veritabanı Yö	SQL Server 2000 Yöneti	William R. Stan	ARKADAS	01.01.2002	11000000,000
WEB Program	Macromedia Fireworks 4	Patti Schulze	SISTEM	01.01.2002	17500000,000
WEB Program	Macromedia Dreamweav	Nolan Hester	SISTEM	01.01.2002	17500000,000
Bestseller	Eğer Öyle Olsayd&	Robert Cowle	0	01.01.2002	14000000,000
Bilgisayar Pro	Adım Adım Microsoft SQ	Rebecca M. R	ARKADAS	01.01.2002	24000000,000
*					

Alan: Fiyat Kriter: >= Deger: 10000000

Siralama: Adı DESC

Fiyat >= 10000000

Temizle Süz

Şekil 3. Programın Çalışması

Örneğin ben, Fiyatı 10 milyon TL' nin üstünde olan kitapların listesini Adlarına göre z den a ya sıralanmış bir şekilde elde ettim. Değerli okurlarım geldik bir makalemizin daha sonuna. DataView nesnesinin özellikle RowFilter tekniğine ilişkin olarak yazdığımız bu makale ile inanıyorum ki yeni fikirler ile donanmışsınızdır. Hepinize mutlu günler dilerim.

DataGrid Denetimi Üzerinde Sayfalama(Paging) İşlemi

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, bir ASP.NET sayfasında yer alan DataGrid kontrolümüzde nasıl sayfalama işlemlerini gerçekleştireceğimizi göreceğiz. Uygulamamız, sql sunucusundaki veritabanımızdan bir tablo ile ilgili bilgileri ekranda gösterecek. Ancak çok sayıda kayıt olduğu için biz bunları, dataGrid kontrolümüzde 10'ar 10'ar göstereceğiz. Konumuzu anlayabilmek için doğrudan kodlama ile işe başlayalım diyorum. Öncelikle VS.NET ile bir ASP.NET Web Application oluşturalım ve WebForm1.aspx sayfamızın adını default.aspx olarak değiştirelim. Şimdi öncelikle bir DataGrid nesnesini sayfamıza yerleştirelim ve hiç bir özelliğini ayarlamayalım. Bunları default.aspx sayfasının html görünümünde elle kodlayacağız. Şu an için DataGrid kontrolümüze ait aspx dataGrid tag'ımızın hali şöyledir.

```
<asp:DataGrid id="dgKitap" style="Z-INDEX: 101; LEFT: 56px; POSITION: absolute; TOP: 56px"
runat="server"></asp:DataGrid>
```

Şimdi, code-behind kısmında yer alacak kodları yazalım. Sql sunucumuza bir bağlantı oluşturacağız, Friends veritabanımızda yer alan Kitaplar tablosundaki satırları bir DataTable nesnesine yükleyip daha sonra dataGrid kontrolümüze bağlayacağız. Bunu sağlayacak olan code-behind kodlarımız ise şu şekilde olacaktır;

```

SqlConnection conFriends;
SqlDataAdapter da;
DataTable dtKitaplar;
public void Baglan()
{
    conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
    da=new SqlDataAdapter("select ID,Kategori,Adi,Yazar,BasimEvi,BasimTarihi,Fiyat from kitaplar order by Adi",conFriends);
    dtKitaplar=new DataTable("Tum Kitaplar");
    da.Fill(dtKitaplar);
}
private void Page_Load(object sender, System.EventArgs e)
{
    Baglan();
    dgKitap.AutoGenerateColumns=false; /* DataGrid kontrolümüzde yer alıcak kolonları kendimiz ayarlayacağımız için bu özelliğe
false değerini aktardık.*/
    dgKitap.DataSource=dtKitaplar; /*DataGrid kontrolümüze veri kaynağı olarak dtKitaplar isimli DataTable nesneminin bellekte
işaret ettiği veri kümesini gösteriyoruz.*/
    dgKitap.DataBind(); /* DataGrid kontrolündeki kolonları (bizim yazdığımız ve ayarladığımız kolonları) veri kaynağındaki ilgili
alanlara bağlıyoruz.*/
}

```

Şimdi sayfamızda yer alan DataGrid tag'ındaki düzenlemelerimizi yapalım. Burada Columns isimli taglar arasında, dataGrid kontrolümüzde görünmesini istediğimiz BoundColumn tipindeki sütunları belirleyeceğimiz tagları yazacağız. Bu sayede DataGrid kontrolüne ait DataBind metodu çağırıldığında, bizim bu taglarda belirttiğimiz alanlar DataGrid kontrolümüzün kolonları olacak şekilde ilgili veri alanlarına bağlanacak. Gelin şimdi buradaki düzenlemeleri gerçekleştirelim. Unutmadan, kendi DataGrid kolonlarınızı ayarlayabilmeniz için AutoGenerateColumns özelliğine false değerini aktarmanız gerekmektedir. Aksi takdirde ayarladığınız kolonların hemen arkasından, otomatik olarak DataTable'da yer alan tüm kolonlar tekrardan gelir. Yaptığımız son güncellemeleri ile DataGrid tag'ımızın yeni hali şu şekildedir.

```

:asp:datagrid id="dgKitap" style="Z-INDEX: 101; LEFT: 24px; POSITION: absolute; TOP: 80px" runat="server"
borderColor="#CC9966" BorderStyle="None" BorderWidth="1px" BackColor="White" CellPadding="4">
    <SelectedItemStyle Font-Bold="True" ForeColor="#663399" BackColor="#FFCC66"></SelectedItemStyle>
    <ItemStyle ForeColor="#330099" BackColor="White"></ItemStyle>
    <HeaderStyle Font-Bold="True" ForeColor="#FFFFCC" BackColor="#990000"></HeaderStyle>
    <FooterStyle ForeColor="#330099" BackColor="#FFFFCC"></FooterStyle>
    <Columns>
        <asp:BoundColumn DataField="ID" ReadOnly="True" HeaderText="No"></asp:BoundColumn>
        <asp:BoundColumn DataField="Kategori" ReadOnly="True" HeaderText="Kategori"></asp:BoundColumn>
        <asp:BoundColumn DataField="Adi" ReadOnly="True" HeaderText="Kitap"></asp:BoundColumn>
        <asp:BoundColumn DataField="Yazar" ReadOnly="True" HeaderText="Yazari"></asp:BoundColumn>
        <asp:BoundColumn DataField="BasimEvi" ReadOnly="True" HeaderText="Yayimlayan"></asp:BoundColumn>
        <asp:BoundColumn DataField="BasimTarihi" ReadOnly="True" HeaderText="Basim
Tarihi"></asp:BoundColumn>
        <asp:BoundColumn DataField="Fiyat" ReadOnly="True" HeaderText="&#199;ikis Fiyati"></asp:BoundColumn>
    </Columns>
    <PagerStyle HorizontalAlign="Center" ForeColor="#330099" BackColor="#FFFFCC"></PagerStyle>
</asp:datagrid>

```

Burada görüldüğü gibi, DataGrid kontrolümüzde görünmesini istediğim tablo alanlarını birer BoundColumn olarak, DataGrid tagları arasına ekledik. Kısaca bahsetmek gerekirse hepsi için, DataField özelliği ile tablodaki hangi alana ait verileri göstereceklerini, HeaderText özelliği ile sütun başlıklarında ne yazacağını, ReadOnly özelliği ile sadece okunabilir alanlar olduklarını belirliyoruz. Bu haliyle uygulamamızı çalıştırsak aşağıdakine benzer bir ekran görüntüsü ile karşılaşırız.

Kitap Listesi

No	Kategorisi	Kitap	Yazari
47	Bestseller	2500 Yıllık Savaş Tarihi	Jhon Keegan
59	Bilgisayar Programlama	Adım Adım Microsoft SQL Server 2000 Programlama	Rebecca M.
33	Bilgisayar Programlama	Adım Adım Microsoft Visual Basic 6 Professional	Michael Hal
13	WEB Programlama	Adım Adım Web Veritabanı Geliştirme	Jim Buyens

Şekil 1. Programın İlk Hali.

Görüldüğü gibi kitap listesi uzayıp gitmektedir. Bizim amacımız bu listeyi 10'arlı gruplar halinde göstermek. Bunun için yapılacak hareket gayet basit gözüksede ince bir teknik kullanmamızı gerektiriyor. Öncelikle dataGrid kontrolümüzün, bir takım özelliklerini belirlemeliyiz. Bu amaçla code-behind kısmında yer alan Page_Load procedure'ünde bir takım değişiklikler yaptık.

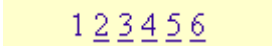
```
private void Page_Load(object sender, System.EventArgs e)
```

```

{
    if(!Page.IsPostBack) /* Sayfa ilk kez yükleniyorsa dataGrid'e ait özellikler belirlensin. Diğer yüklemelerde tekrardan bu işlemler
yapılmasın istediğimiz için...*/
    {
        dgKitap.AllowPaging=true; /* DataGrid kontrolümüzde sayfalama yapılabilmesini sağlıyoruz.*/
        dgKitap.PagerStyle.Mode=PagerMode.NumericPages; /* Sayfalama sistemi sayısal olucak. Yani 1 den başlayıp kaç kayıtlık
sayfa oluştuysa o kadar sayıda bir buton dizesi dataGrid kontrolünün en altında yer alacak.*/
        dgKitap.AutoGenerateColumns=false; /* DataGrid kontrolümüzde yer alacak kolonları kendimiz ayarlayacağımız için bu
özellikle false değerini aktadık.*/
    }
    Baglan();
    dgKitap.DataSource=dtKitaplar; /*DataGrid kontrolümüze veri kaynağı olarak dtKitaplar isimli DataTable nesnemizin bellekte
işaret ettiği veri kümesini gösteriyoruz.*/
    dgKitap.DataBind(); /* DataGrid kontrolündeki kolonları (bizim yazdığımız ve ayarladığımız kolonları) veri kaynağındaki ilgili
alanlara bağlıyoruz.*/
}

```

Şimdi kodumuzu yeniden çalıştırsak bu kez DataGrid kontrolümüzün alt kısmında sayfa linklerinin oluştuğunu görürüz.



Şekil 2. Sayfa Linkleri

Ancak bu linklerden herhangi birine bastığımızda ilgili sayfaya gidemediğimizi aynı sayfanın gerisin geriye geldiğini görürüz. İşte pek çoğumuzun zorlandığı ve gözden kaçırdığı teknik burada kendini gösterir. Aslında her şey yolunda gözükmemektedir ve sistem çalışmalıdır. Ama çalışmamaktadır. Yapacağımız bu sayfalama işlemini gerçekleştirecek bir metod yazmak ve son olarakta bu metodu DataGrid tag'ına yazacağımız OnPageIndexChanges olay procedure'ü ile ilişkilendirmektir. OnPageIndexChanges olayı DataGrid kontrolünde yer alan sayfalama linklerinden birine basıldığında çalışacak kodları içerir. Bu durumda DataGrid tag'ımızın son hali aşağıdaki gibi olur.


```

<asp:datagrid id="dgKitap" style="Z-INDEX: 101; LEFT: 24px; POSITION: absolute; TOP: 80px" runat="server"
BorderColor="#CC9966" BorderStyle="None" BorderWidth="1px" BackColor="White" CellPadding="4"
OnPageIndexChanged="Sayfa_Guncelle">
    <SelectedItemStyle Font-Bold="True" ForeColor="#663399" BackColor="#FFCC66"></SelectedItemStyle>
    <ItemStyle ForeColor="#330099" BackColor="White"></ItemStyle>
    <HeaderStyle Font-Bold="True" ForeColor="#FFFFCC" BackColor="#990000"></HeaderStyle>
    <FooterStyle ForeColor="#330099" BackColor="#FFFFCC"></FooterStyle>
    <Columns>
        <asp:BoundColumn DataField="ID" ReadOnly="True" HeaderText="No"></asp:BoundColumn>
        <asp:BoundColumn DataField="Kategori" ReadOnly="True" HeaderText="Kategorisi"></asp:BoundColumn>
        <asp:BoundColumn DataField="Adi" ReadOnly="True" HeaderText="Kitap"></asp:BoundColumn>
        <asp:BoundColumn DataField="Yazar" ReadOnly="True" HeaderText="Yazari"></asp:BoundColumn>
        <asp:BoundColumn DataField="BasimEvi" ReadOnly="True" HeaderText="Yayimlayan"></asp:BoundColumn>
        <asp:BoundColumn DataField="BasimTarihi" ReadOnly="True" HeaderText="Basim
Tarihi"></asp:BoundColumn>
        <asp:BoundColumn DataField="Fiyat" ReadOnly="True" HeaderText="Fiyati"></asp:BoundColumn>
    </Columns>
    <PagerStyle HorizontalAlign="Center" ForeColor="#330099" BackColor="#FFFFCC"></PagerStyle>
</asp:datagrid>

```

Şimdide code_behind kısmında Sayfa_Guncelle metodumuzu ekleyelim.

```
Public void Sayfa_Guncelle(object sender , DataGridPageChangedEventArgs e)
```

```
{
```

```
    dgKitap.CurrentPageIndex=e.NewPageIndex; /* İşte olayı bitiren hamle. CurrentPageIndex özelliğine basılan linkin temsil ettiği
sayfanın index nosu aktarılıyor. Böylece belirtilen sayfaya geçilmiş oluyor. Ancak iş bununla bitmiyor. Veritabanından verilerin
tekrardan yüklenmesi ve dataGrid kontrolümüze bağlanması gerekli.*/
```

```
    Baglan();
```

```
    dgKitap.DataSource=dtKitaplar;
```

```
    dgKitap.DataBind();
```

```
}
```

Şimdi uygulamamızı çalıştırsak eğer, sayfalar arasında rahatça gezebildiğimizi görürüz. Geldik bir makalemizin daha sonuna, bir sonraki makalemizde, yine DataGrid kontrolünü inceleyeceğiz. Bu defa, kolonlar üzerinden sıralama işlemlerinin nasıl yapıldığını incelemeye çalışacağız. Umuyorumki hepiniz için faydalı bir makale olmuştur. Hepinize mutlu günler dilerim.

DataGrid Denetimi Üzerinde Sıralama(Sorting) İşlemi

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, bir Web Sayfası üzerinde yer alan DataGrid kontrolü üzerinde tıklanan kolon başlığına göre sıralama işleminin manuel olarak nasıl yapılacağını işleyeceğiz. Konu teknik ağırlığa sahip olduğu için hemen kodlara geçmek istiyorum.

Uygulamamız , C# ile yazılmış bir Web Application. Bir adet herhangi bir özelliği belirlenmemiş DataGrid kontrolü içermekte. Aspx sayfamızın kodlarına göz atacak olursak, DataBound tagları içerisinde yer alan SortExpression ifadeleri ve DataGrid tagında yer alan, OnSortCommand ifadesi bizim anahtar terimlerimizdir. SortExpression ifadesi, kolon başlığına tıklandığında ilgili veri kümesinin hangi alan adını göz önüne alacağını belirlemek için kullanılır. OnSortCommand değeri ise, SortExpression ifadesinin işlenerek sıralamanın yapılacağı kodları içeren procedure adına işaret etmektedir. Bu bilgiler ışığında izleyeceğimiz yol şudur;

1- DataBound tagları içinde SortExpression değerlerini belirlemek.
2- DataGrid tagı içinde, OnSortCommand olayı için metodu belirlemek.
3- OnSortCommand olayı için ilgili metodu geliştirmek.

Şimdi öncelikle default.aspx sayfamızın içeriğine bir bakalım.

```
<asp:DataGrid id="DataGrid1" style="Z-INDEX: 101; LEFT: 24px; POSITION: absolute; TOP: 32px" runat="server"
OnSortCommand="Siralala" BorderColor="#CC9966" BorderStyle="None" BorderWidth="1px" BackColor="White">
<SelectedItemStyle Font-Bold="True" ForeColor="#663399" BackColor="#FFCC66"></SelectedItemStyle>
<ItemStyle ForeColor="#330099" BackColor="White"></ItemStyle>
<HeaderStyle Font-Bold="True" ForeColor="#FFFFCC" BackColor="#990000"></HeaderStyle>
<FooterStyle ForeColor="#330099" BackColor="#FFFFCC"></FooterStyle>
<Columns>
<asp:BoundColumn DataField="ID" SortExpression="ID" ReadOnly="True" HeaderText="ID"></asp:BoundColumn>
<asp:BoundColumn DataField="Adi" SortExpression="Adi" ReadOnly="True" HeaderText="Kitap"></asp:BoundColumn>
<asp:BoundColumn DataField="Yazar" SortExpression="Yazar" ReadOnly="True" HeaderText="Yazar(lar)"></asp:BoundColumn>
<asp:BoundColumn DataField="BasimEvi" SortExpression="BasimEvi" ReadOnly="True" HeaderText="Yayimlayan"></asp:BoundColumn>
<asp:BoundColumn DataField="Fiyat" SortExpression="Fiyat" ReadOnly="True" HeaderText="&#199;ikis Fiyati"></asp:BoundColumn>
</Columns>
<PagerStyle HorizontalAlign="Center" ForeColor="#330099" BackColor="#FFFFCC"></PagerStyle>
</asp:DataGrid>
```

Şimdi ise code-behind kısmında yer alan default.aspx.cs dosyamızın içeriğine bir bakalım.

```
SqlConnection conFriends;
SqlDataAdapter da;
DataTable dtKitaplar;
DataView dvKitaplar;

/* Sql sunucumuzda yer alan Friends isimli veritabanına bağlanıyoruz. Buradan Kitaplar isimli tablodaki verileri SqlDataAdapter nesnemiz ile alıp dataTable nesnemizin bellekte işaret ettiği yere aktarıyoruz. Daha sonra ise dataTable nesnemizin defaultView metodunu kullanarak, dataView nesnemizi varsayılan tablo görünümü ile dolduruyoruz. Eğer sayfalarımızda sadece görsel amaçlı dataGrid'ler kullanacaksak yada başka bir deyişle bilgilendirme amaçlı veri kümelerini sunacaksak DataView nesnelerini kullanmak performans açısından fayda sağlayacaktır.*/

public void Baglan()
{
    conFriends =new SqlConnection("Data source=localhost;integrated security=sspi;initial catalog=Friends");
    da=new SqlDataAdapter("Select ID,Adi,Yazar,BasimEvi,Fiyat From Kitaplar",conFriends);
    dtKitaplar=new DataTable("Kitap Listesi");
    da.Fill(dtKitaplar);
}
```

```

dvKitaplar=dtKitaplar.DefaultView;
DataGrid1.AutoGenerateColumns=false; /* DataGrid nesneminin içereceği kolonları kendimiz belirlemek istediğimizden
AutoGenerateColumns özelliğine false değerini atadık.*/
DataGrid1.AllowSorting=true; /* AllowSorting özelliğine true değerini aktardığımızda, DataGrid'in başlık kısmında yer alan kolon
isimlerine tıkladığımızda bu alanlara göre sıralama yapabilmelerini sağlamış oluyoruz. */
}
/* Sırala isimli metodumuz, DataGrid tagında OnSortCommand için belirttiğimiz metoddur. Bu metod ile , bir kolon başlığına
tıklandığında yapılacak sıralama işlemlerini belirtiyoruz. Bu metod, DataGridSortCommandEventArgs tipinde bir parametre
almaktadır. Bu parametremizin SortExpression değeri, tıklanan kolon başlığının dataGrid tagında,bu alan ile ilgili olan DataBound
sekmesinde yer alan SortExpression ifadesine atanan değerdir. Biz bu değeri alarak DataView nesneminin Sort metoduna
gönderiyoruz. Böylece DataView nesnesinin bellekte işaret ettiği veri kümesini e.SortExpression özelliğinin değerine göre yani
seçilen alana göre sıralatmış oluyoruz. Daha sonra ise yaptığımız işlem DataGrid kontrolümüzü tekrar bu veri kümesine bağlamak
oluyor.*/
public void Sırala(object sender,DataGridSortCommandEventArgs e)
{
    lblSıralamaKriteri.Text="Sıralama Kriteri : "+e.SortExpression.ToString();
    dvKitaplar.Sort=e.SortExpression;
    DataGrid1.DataSource=dvKitaplar;
    DataGrid1.DataBind();
}
private void Page_Load(object sender, System.EventArgs e)
{
    Baglan();
    DataGrid1.DataSource=dvKitaplar;
    DataGrid1.DataBind();
}

```

Şimdi uygulamamızı çalıştıralım ve kolon başlıklarına tıklayarak sonuçları izleyelim. İşte örnek ekran görüntüleri.

Sıralama Kriteri : Adı

ID	Kitap	Yazar(lar)
47	2500 Yıllık Savaş Tarihi	Jhon Kee
59	Adım Adım Microsoft SQL Server 2000 Programlama	Rebecca
33	Adım Adım Microsoft Visual Basic 6 Professional	Michael F
13	Adım Adım Web Veritabanı Geliştirme	Jim Buye
64	Adım Adım XML	Michael J
12	Analiz, tasarım ve Uygulamalı Sistem Yönetimi	Prof.Dr.F
9	ASP 3.0 Active Server Pages Web Programcılığı Temel Başlangıç Kılavuzu	Nicholas
8	ASP ile E-Ticaret Programcılığı	Stephen T
10	ASP ile web programcılığı ve elektronik ticaret.	Zafer Der
51	Başarı İçin Stratejiler	Jim Dorn
40	Bedavacının Web Sitesi Tasarım Kılavuzu Bir Kuruş Bile Harcamadan Web Sitesi Sahibi Olun	Mehmet I
41	Delphi 5'e Bakış	Ruhvar

*Şekil 1. Kitap Adına göre sıralanmış hali.***Sıralama Kriteri : ID**

ID	Kitap	Yazar(lar)
1	Delphi 5'e Bakış	Ruhvar
3	Delphi 5 Uygulama Geliştirme Kılavuzu	Marco C
4	Delphi 5 Kullanım Kılavuzu	Dr. Cahi
5	Microsoft Visual Basic 6.0 Geliştirmek Ustalaşma Dizisi	Teresa C Parrent
6	Visual Basic 6 Temel Başlangıç Kılavuzu	Greg Pe
7	Microsoft Visual Basic 6 Temel Kullanım Kılavuzu Herkes İçin!	Faruk Ç
8	ASP ile E-Ticaret Programcılığı	Stephen
9	ASP 3.0 Active Server Pages Web Programcılığı Temel Başlangıç Kılavuzu	Nichola
10	ASP ile web programcılığı ve elektronik ticaret	Zafer D

Şekil 2. ID alanına göre sıralanmış hali.

Sıralama Kriteri : Yazar

ID	Kitap	Yazar(lar)
42	Kim Güler Bilgisayarlara? "Bilgisayarlar Akıllı Değildir, Sadece Kendilerini Öyle Sanırlar"	Can Uğur
14	Macromedia Flash 5 actionscript yaratıcı web animasyonları	Derek Fra
24	Matematik Analiz 3	Doç. Dr. C
25	Matematik Analiz 4	Doç. Dr. C

Şekil 3. Yazar adına göre sıralanmış hali.

Geldik bir makalemizin daha sonuna, bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

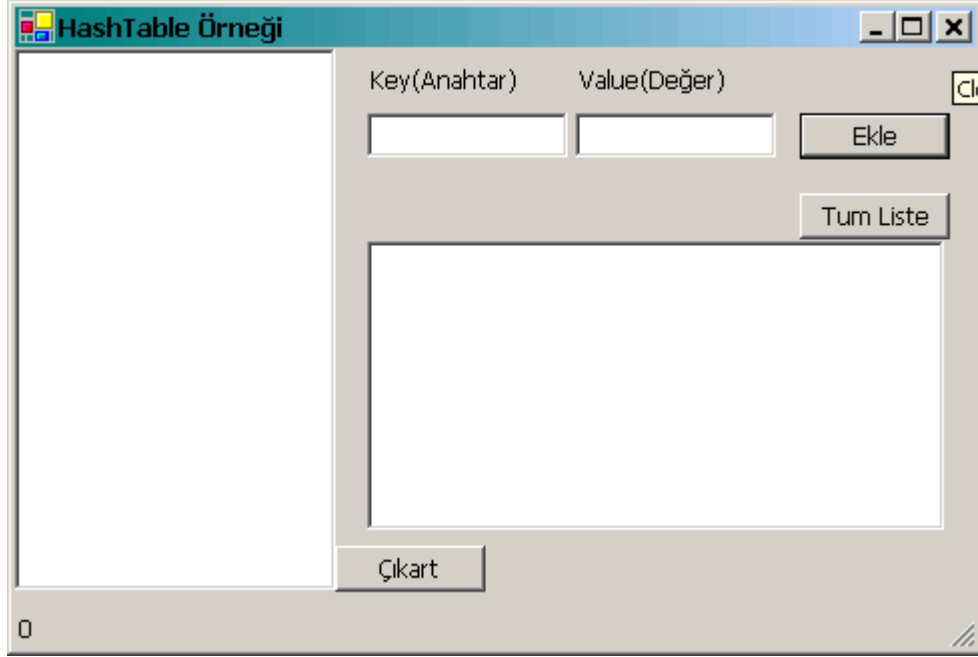
HashTable Koleksiyon Sınıfı

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde HashTable koleksiyon sınıfını incelemeye çalışacağız. Bildiğiniz gibi Koleksiyonlar System.Collections namespace'inde yer almakta olup, birbirlerinin aynı veya birbirlerinden farklı veri tiplerinin bir arada tutulmasını sağlayan diziler oluşturmamıza imkan sağlamaktadırlar. Pek çok koleksiyon sınıfı vardır. Bugün bu koleksiyon sınıflarından birisi olan HashTable koleksiyon sınıfını inceleyeceğiz. HashTable koleksiyon sınıfında veriler key-value dediğimiz anahtar-değer çiftleri şeklinde tutulmaktadırlar. Tüm koleksiyon sınıflarının ortak özelliği barındırdıkları verileri object tipinde olmalıdır. Bu nedenle, HashTable'lardada key ve value değerleri herhangi bir veri tipinde olabilirler. Temel olarak bunların her biri birer DictionaryEntry nesnesidir. Bahsetmiş olduğumuz key-value çiftleri hash tablosu adı verilen bir tabloda saklanırlar. Bu değer çiftlerine erişmek için kullanılan bir takım karmaşık kodlar vardır.

Key değerleri tektir ve değiştirilemezler. Yani bir key-value çiftini koleksiyonumuza eklediğimizde, bu değer çiftinin value değerini değiştirebilirken, key değerini değiştiremeyiz. Ayrıca key değerleri benzersiz olduklarında tam anlamıyla birer anahtar alan vazifesi görürler. Diğer yandan value değerline null değerler atayabilirken, anahtar alan niteliğindeki Key değerlerine null değerler atayamayız. Şayet uygulamamızda varolan bir Key değerini eklemek istersek ArgumentException istisnası ile karşılaşırız.

HashTable koleksiyonu verilere hızlı bir biçimde ulaşmamızı sağlayan bir kodlama yapısına sahiptir. Bu nedenle özellikle arama maliyetlerini düşürdüğü için tercih edilmektedir. Şimdi konuyu daha iyi pekiştirebilmek amacıyla, hemen basit bir uygulama geliştirelim. Uygulamamızda, bir HashTable koleksiyonuna key-value çiftleri ekliyecek, belirtilen key'in sahip olduğu değere bakılacak, tüm HashTable'in içerdiği key-value çiftleri listelenecek, eleman çiftlerini HashTable'dan çıkartacak vb... işlemler gerçekleştireceğiz. Form tasarımı ben aşağıdaki şekildeki gibi yaptım. Temel olarak teknik terimlerin türkçe karşılığına dair minik bir sözüğü bir HashTable olarak tasarlayacağız.



1. Form Tasarımımız.

Şimdi kodlarımıza bir göz atalım.

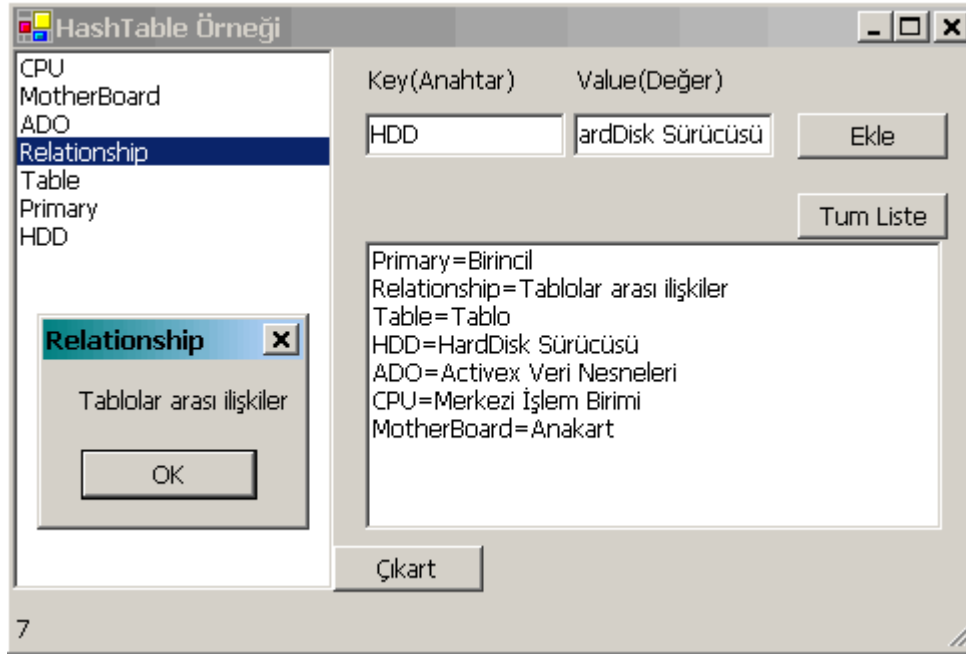
```
System.Collections.Hashtable htTeknikSozluk; /* HashTable koleksiyon nesnemizi tanımlıyoruz.*/
private void Form1_Load(object sender, System.EventArgs e)
{
    htTeknikSozluk=new System.Collections.Hashtable(); /* HashTable nesnemizi oluşturuyoruz.*/
    stbDurum.Text=htTeknikSozluk.Count.ToString(); /* HashTable'ımızdaki eleman sayısını Count özelliği ile öğreniyoruz.*/
}
private void btnEkle_Click(object sender, System.EventArgs e)
{
    try
    {
        htTeknikSozluk.Add(txtKey.Text,txtValue.Text);/* HashTable'ımıza key-value çifti ekleyebilmek için Add metodu
        kullanılıyor.*/
        lstAnahtar.Items.Add(txtKey.Text);
        stbDurum.Text=htTeknikSozluk.Count.ToString();
    }
    catch(System.ArgumentException) /* Eğer var olan bir key'i tekrar eklemeye çalışırsak bu durumda ArgumentException istisnası
    fırlatılacaktır. Bu durumda, belirtilen key-value çifti HashTable koleksiyonuna eklenmez. Bu durumu kullanıcıya bildiriyoruz.*/
    {
        stbDurum.Text=txtKey.Text+" Zaten HashTable Koleksiyonunda Mevcut!";
    }
}
```

```

}
private void IstAnahtar_DoubleClick(object sender, System.EventArgs e)
{
    string deger;
    deger=htTeknikSozluk[IstAnahtar.SelectedItem.ToString()].ToString(); /* HashTable'daki bir değere ulaşmak için, köşeli
    parantezler arasında aranacak key değerini giriyoruz. Sonucu bir string değişkenine aktarıyoruz.*/
    MessageBox.Show(deger,IstAnahtar.SelectedItem.ToString());
}
private void btnSil_Click(object sender, System.EventArgs e)
{
    if(htTeknikSozluk.Count==0)
    {
        stbDurum.Text="Çıkarılabilecek hiç bir eleman yok";
    }
    else if(IstAnahtar.SelectedIndex== -1)
    {
        stbDurum.Text="Listeden bir eleman seçmelisiniz";
    }
    else
    {
        htTeknikSozluk.Remove(IstAnahtar.SelectedItem.ToString()); /* Bir HashTable'dan bir nesneyi çıkartmak için, Remove
        metodu kullanılır. Bu metod parametre olarak çıkartılmak istenen değer çiftinin key değerini alır.*/
        IstAnahtar.Items.Remove(IstAnahtar.SelectedItem);
        stbDurum.Text="Çıkarıldı";
        stbDurum.Text=htTeknikSozluk.Count.ToString();
    }
}
private void btnTumu_Click(object sender, System.EventArgs e)
{
    IstTumListe.Items.Clear();
    /* Aşağıdaki satırlarda, bir HashTable koleksiyonu içinde yer alan tüm elemanlara nasıl erişildiğini görmekteyiz. Keys metodu ile
    HashTable koleksiyonumuzda yer alan tüm anahtar değerlerini (key'leri), ICollection arayüzü(interface) türünden bir nesneye
    atıyoruz. Foreach döngümüz ile bu nesne içindeki her bir anahtarı, HashTable koleksiyonunda bulabiliyoruz.*/
    ICollection anahtar=htTeknikSozluk.Keys;
    foreach(string a in anahtar)
    {
        IstTumListe.Items.Add(a+"="+htTeknikSozluk[a].ToString());
    }
}

```

Şimdi uygulamamızı çalıştırıp deneyelim.



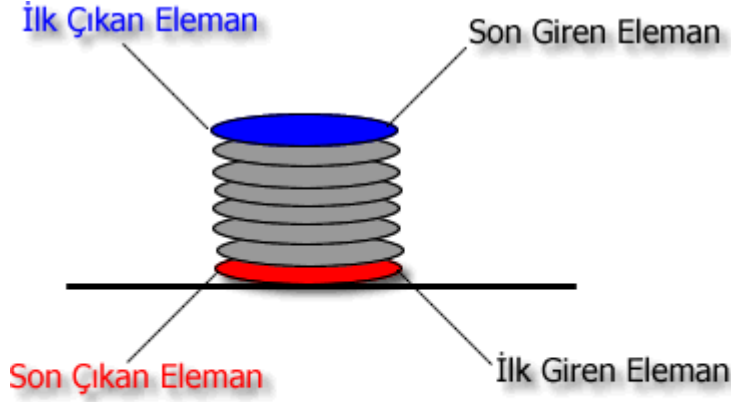
2. Programın Çalışmasının sonucu.

Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Stack ve Queue Koleksiyon Sınıfı

Değerli Okurlarım, Merhabalar.

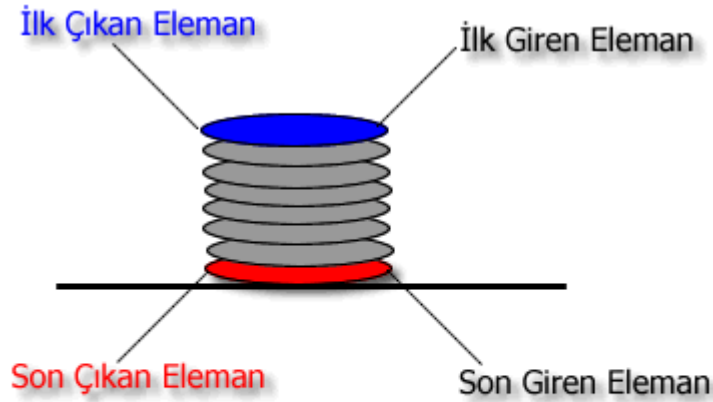
Bugünkü makalemizde Stack ve Queue koleksiyon sınıflarını incelemeye çalışacağız. Bir önceki makalemizde bildiğiniz gibi, HashTable koleksiyon sınıfını incelemiştik. Stack ve Queue koleksiyonlarında, System.Collections isim alanında yer alan ve ortak koleksiyon özelliklerine sahip sınıflardır. Stack ve Queue koleksiyonları, her koleksiyon sınıfında olduğu gibi, elemanlarını nesne (object) tipinde tutmaktadırlar. Bu koleksiyonların özelliği giren-çıkan eleman prensibleri üzerine çalışmalarıdır. Stack koleksiyon sınıfı, LIFO adı verilen, Last In First Out(Son giren ilk çıkar) prensibine göre çalışırken, Queue koleksiyon sınıfı ise FIFO yani First In First Out(ilk giren ilk çıkar) prensibine göre çalışır.Konuyu daha iyi anlayabilmek için aşağıdaki şekilleri göz önüne alalım.



LIFO (Last In First Out-Son Giren İlk Çıkar) Stack Koleksiyon Sınıfı

Şekil 1. Stack Koleksiyon Sınıfının Çalışma Yapısı

Görüldüğü gibi, Stack koleksiyonunda yer alan elemanlardan son girene ulaşmak oldukça kolaydır. Oysaki ilk girdiğimiz elemana ulaşmak için, bu elemanın üstünde yer alan diğer tüm elemanları silmemiz gerekmektedir. Queue koleksiyon sınıfına gelince;



FIFO (First In First Out-İlk Giren İlk Çıkar) Queue Koleksiyon Sınıfı

Şekil 2. Queue Koleksiyon Sınıfının Çalışma Yapısı

Görüldüğü gibi Queue koleksiyon sınıfında elemanlar koleksiyona arkadan katılırlar ve ilk giren eleman kuyruktan ilk çıkan eleman olur. Stack ve Queue farklı yapılarda tasarlandıkları için elemanlarına farklı metodlar ile ulaşılmaktadır. Stack koleksiyon sınıfında, en son giren elemanı elde etmek için Pop metodu kullanılır. Koleksiyona bir eleman eklerken Push metodu kullanılır. Elbette eklenen eleman en son elemandır ve Pop metodu çağırıldığında elde edilecek olan ilk eleman halini alır. Ancak Pop metodu son giren elemanı verirken bu elemanı koleksiyondan siler. İşte bunun önüne geçen metod Peek metodudur. Şimdi diyebilirsiniz ki maden son giren elemanı siliyor Pop metodu o zaman niye kullanıyoruz. Hatırlarsanız, Stack koleksiyonunda, ilk giren elemanı elde etmek için bu elemanın üstünde yer alan tüm elemanları silmemiz gerektiğini söylemiştik. İşte bir döngü yapısında Pop metodu

kullanıldığında, ilk giren elemana kadar inebiliriz. Tabi diğer elemanları kaybettikten sonra bunun çok büyük önem taşıyan bir eleman olmasını isteyebiliriz.

Gelelim Queue koleksiyon sınıfının metodlarına. Dequeue metodu ile koleksiyona ilk giren elemanı elde ederiz. Ve bunu yaptığımız anda eleman silinir. Nitekim dequeue metodu pop metodu gibi çalışır. Koleksiyona eleman eklemek için ise, enqueue metodu kullanılır. İlk giren elemanı elde etmek ve silinmemesini sağlamak istiyorsak yine stack koleksiyon sınıfında olduğu gibi, Peek metodunu kullanırız. Bu koleksiyonların en güzel yanlarından birisi size leman sayısını belirtmediğiniz takdirde koleksiyonun boyutunu otomatik olarak kendilerinin ayarlamalarıdır. Stack koleksiyon sınıfı, varsayılan olarak 10 elemanlı bir koleksiyon dizisi oluşturur. (Eğer biz eleman sayısını yapıcı metodumuzda belirtmez isek). Eğer eleman sayısı 10'u geçerse, koleksiyon dizisinin boyutu otomatik olarak iki katına çıkar. Aynı prensib queue koleksiyon sınıfı içinde geçerli olmakla birlikte, queue koleksiyonu için varsayılan dizi boyutu 32 elemanlı bir dizidir. Şimdi dilerseniz, basit bir console uygulaması ile bu konuyu anlamaya çalışalım.

```
using System;
using System.Collections; /* Uygulamalarımızda koleksiyon sınıflarını kullanabilmek için Collections isim uzayını kullanmamız gerekir.*/
namespace StackSample1
{
    class Class1
    {
        static void Main(string[] args)
        {
            Stack stc=new Stack(4); /* 4 elemanlı bir Stack koleksiyonu oluşturduk.*/
            stc.Push("Burak"); /*Eleman eklemek için Push metodu kullanılıyor.*/
            stc.Push("Selim");
            stc.Push("ŞENYURT");
            stc.Push(27);
            stc.Push(true);
            Console.WriteLine("Çıkan eleman {0}",stc.Pop().ToString());/* Pop metodu son giren(kalan) elemanı verirken, aynı zamanda bu elemanı koleksiyon dizisinden siler.*/
            Console.WriteLine("Çıkan eleman {0}",stc.Pop().ToString());
            Console.WriteLine("Çıkan eleman {0}",stc.Pop().ToString());
            Console.WriteLine("-----");
            IEnumerator dizi=stc.GetEnumerator(); /* Koleksiyonun elemanlarını IEnumerator arayüzünden bir nesneye aktarıyoruz.*/
            while(dizi.MoveNext()) /* dizi nesnesinde okunacak bir sonraki eleman var olduğu sürece işleyecek bir döngü.*/
            {
                Console.WriteLine("Güncel eleman {0}",dizi.Current.ToString()); /* Current metodu ile dizi nesnesinde yer alan güncel elemanı elde ediyoruz. Bu döngüyü çalıştırdığımızda sadece iki elemanın dizide olduğunu görürüz. Pop metodu sağolsun.*/
            }
            Console.WriteLine("-----");
        }
    }
}
```

```

        Console.WriteLine("En üstteki eleman {0}",stc.Peek()); /* Peek metodu son giren elemanı veya en üste kalan elemanı
verirken bu elemanı koleksiyondan silmez.*/
        dizi=stc.GetEnumerator();
        while(dizi.MoveNext())
        {
            Console.WriteLine("Güncel eleman {0}",dizi.Current.ToString()); /* Bu durumda yine iki eleman verildiğini Peek
metodu ile elde edilen elemanın koleksiyondan silinmediğini görürüz.*/
        }
    }
}
}
}

```

```

D:\vs samples\StackSample1\
Çıkan eleman True
Çıkan eleman 27
Çıkan eleman SENYURT
-----
Güncel eleman Selim
Güncel eleman Burak
-----
En üstteki eleman Selim
Güncel eleman Selim
Güncel eleman Burak
Press any key to continue

```

Şekil 3. Stack ile ilgili programın çalışmasının sonucu.

Queue örneğimiz ise aynı kodlardan oluşuyor sadece metod isimleri farklı.

```

using System;
using System.Collections;
namespace QueueSample1
{
    class Class1
    {
        static void Main(string[] args)
        {
            Queue qu=new Queue(4);
            qu.Enqueue("Burak"); /*Eleman eklemek için Enqueue metodu kullanılıyor.*/
            qu.Enqueue("Selim");
            qu.Enqueue("ŞENYURT");
            qu.Enqueue(27);
            qu.Enqueue(true);
            Console.WriteLine("Çıkan eleman {0}",qu.Dequeue().ToString());/* Dequeue metodu ilk giren(en alttaki) elemanı

```

verirken, aynı zamanda bu elemanı koleksiyon dizisinden siler.*/

```
Console.WriteLine("Çıkan eleman {0}",qu.Dequeue().ToString());
```

```
Console.WriteLine("Çıkan eleman {0}",qu.Dequeue().ToString());
```

```
Console.WriteLine("-----");
```

IEnumerator dizi=qu.GetEnumerator(); /* Koleksiyonun elemanlarını IEnumerator arayüzünden bir nesneye aktarıyoruz.*/

```
while(dizi.MoveNext()) /* dizi nesnesinde okunacak bir sonraki eleman var olduğu sürece işleyecek bir döngü.*/
```

```
{
```

```
    Console.WriteLine("Güncel eleman {0}",dizi.Current.ToString()); /* Current metodu ile dizi nesnesinde yer alan
```

güncel elemanı elde ediyoruz. Bu döngüyü çalıştırdığımızda sadece iki elemanın dizide olduğunu görürüz. Dequeue metodu sağolsun.*/

```
}
```

```
Console.WriteLine("-----");
```

Console.WriteLine("En altta kalan eleman {0}",qu.Peek()); /* Peek metodu son giren elemanı veya en üste kalan elemanı verirken bu elemanı koleksiyondan silmez.*/

```
dizi=qu.GetEnumerator();
```

```
while(dizi.MoveNext())
```

```
{
```

```
    Console.WriteLine("Güncel eleman {0}",dizi.Current.ToString()); /* Bu durumda yine iki eleman verildiğini Peek
```

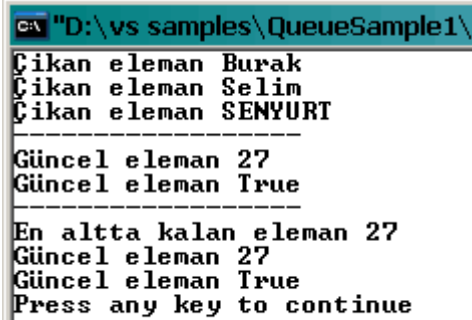
metodu ile elde edilen elemanın koleksiyondan silinmediğini görürüz.*/

```
}
```

```
}
```

```
}
```

```
}
```



```
C:\D:\vs samples\QueueSample1\
Çıkan eleman Burak
Çıkan eleman Selim
Çıkan eleman SENYURT
-----
Güncel eleman 27
Güncel eleman True
-----
En altta kalan eleman 27
Güncel eleman 27
Güncel eleman True
Press any key to continue
```

Şekil 4. Queue ile ilgili programın çalışmasının sonucu.

Geldik bir makalemizin daha sonuna. Umuyorumki sizlere faydalı olabilecek bilgiler sunabilmişimdir. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

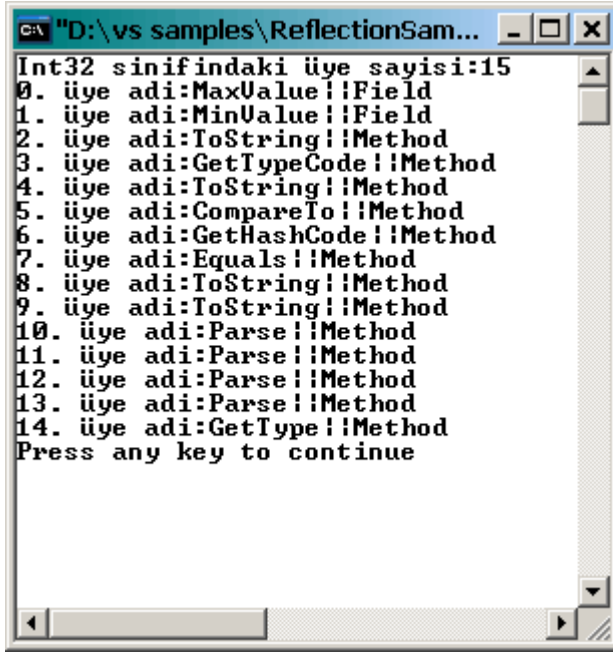
Reflection Sınıfı İle Tiplerin Sırrı Ortaya Çıkıyor

Değerli Okurlarım, Merhabalar.

Hiç .NET 'te yer alan bir tipinin üyelerini öğrenebilmek istediniz mi? Örneğin var olan bir .NET sınıfının veya sizin kendi yazmış olduğunuz yada bir başkasının yazdığı sınıfa ait tüm üyelerin neler olduğuna programatik olarak bakmak istediniz mi? İşte bugünkü makalemizin konusu bu. Herhangi bir tipe (type) ait üyelerin neler olduğunu anlayabilmek. Bu amaçla, Reflection isim uzayını ve bu uzaya ait sınıfları kullanacağız. Bildiğiniz gibi .NET 'te kullanılan her şey bir tipe aittir. Yani herşeyin bir tipi vardır. Üyelerini öğrenmek isteğimiz bir tipi öncelikle bir Type değişkeni olarak alırız. (Yani tipin tipini alırız. Bu nedenle ben bu tekniğe Tip-i-Tip adını verdim). Bu noktadan sonra Reflection uzayına ait sınıfları ve metodlarını kullanarak ilgili tipe ait tüm bilgileri edinebiliriz. Küçük bir Console uygulaması ile konuyu daha iyi anlamaya çalışalım. Bu örneğimizde, System.Int32 sınıfına ait üyelerin bir listesini alacağız. İşte kodlarımız;

```
using System;
namespace ReflectionSample1
{
    class Class1
    {
        static void Main(string[] args)
        {
            Type tipimiz=Type.GetType("System.Int32");/* Öncelikle String sınıfının tipini öğreniyoruz. */
            System.Reflection.MemberInfo[] tipUyeleri=tipimiz.GetMembers(); /* Bu satır ile, System.String tipi içinde yer alana
            üyelerin listesini Reflection uzayında yer alan, MemberInfo sınıfı tipinden bir diziye aktarıyoruz. */
            Console.WriteLine(tipimiz.Name.ToString()+" sınıfındaki üye sayısı:"+tipUyeleri.Length.ToString());/* Length özelliği,
            MemeberInfo tipindeki dizimizde yer alan üyelerin sayısını, (dolayısıyla System.String sınıfı içinde yer alan üyelerin sayısını)
            veriyor.*/
            /* İzleyen döngü ile, MemberInfo dizisinde yer alan üyelerin birtakım bilgilerini ekrana yazıyoruz.*/
            for(int i=0;i<tipUyeleri.Length;++i)
            {
                Console.WriteLine(i.ToString()+". üye
                adı:"+tipUyeleri[i].Name.ToString()+"|" +tipUyeleri[i].MemberType.ToString()); /* Name özelliği üyenin adını verirken,
                MemberType özelliği ile, üyenin tipini alıyoruz. Bu üye tipi metod, özellik, yapıcı metod vb... dir.*/
            }
        }
    }
}
```

Uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 1. System.Int32 sınıfının üyeleri.

Üye listesini incelediğimizde 15 üyenin olduğunu görürüz. Metodlar, alanlar vardır. Ayrıca dikkat ederseniz, Parse , ToString metodları birden fazla defa geçmektedir. Bunun nedeni bu metodların overload (aşırı yüklenmiş) versiyonlara sahip olmasıdır. Kodları incelediğimizde, System.Int32 sınıfına ait tipleri GetMembers metodu ile, System.Reflection uzayında yer alan MemberInfo sınıfı tipinden bir diziye aldığımızı görürüz. İşte olayın önemli kodları bunlardan oluşmaktadır. MemberInfo dışında kullanabileceğimiz başka Reflection sınıfları da vardır. Bunlar;

ConstructorInfo	Tip e ait yapıcı metod üyelerini ve bu üyelere ait bilgilerini içerir.
EventInfo	Tip e ait olayları ve bu olaylara ait bilgileri içerir.
MethodInfo	Tip e ait metodları ve bu metodlara ait bilgileri içerir.
FieldInfo	Tip içinde kullanılan alanları ve bu alanlara ilişkin bilgileri içerir.
ParameterInfo	Tip içinde kullanılan parametreleri ve bu parametrelere ait bilgileri içerir.
PropertyInfo	Tip içinde kullanılan özellikleri ve bu özelliklere ait bilgileri içerir.

Tablo 1. Reflection Uzayının Diğer Kullanışlı Sınıfları

Şimdi bu sınıflara ait örneklerimizi inceleyelim. Bu kez bir DataTable sınıfının üyelerini inceleyeceğiz. Örnek olarak, sadece olaylarını ve bu olaylara ilişkin özelliklerini elde etmeye çalışalım. Bu örneğimizde, yukarıda bahsettiğimiz tip-i-tip tekniğini biraz değiştireceğiz. Nitekim bu tekniği uygulamamız halinde bir hata mesajı alırız. Bunun önüne geçmek için, bir DataTable örneği (instance) oluşturup bu örneğin tipinden hareket edeceğiz. Dilerseniz hemen kodlarımıza geçelim.

```

using System;
namespace ReflectionSample2
{
    class Class1
    {
        static void Main(string[] args)
        {
            System.Data.DataTable dt=new System.Data.DataTable(); /* Bir DataTable örneği(instance) yaratıyoruz.*/
            Type tipimiz=dt.GetType(); /* DataTable örneğimizin GetType metodunu kullanarak, bu örneğin dolayısıyla DataTable
sınıfının tipini elde ediyoruz. */
            System.Reflection.MethodInfo[] tipMetodlari=tipimiz.GetMethods(); /* Bu kez sadece metodları incelemek
istediğimizden, GetMethods metodunu kullanıyor ve sonuçları, MethodInfo sınıfı tipinden bir diziye aktarıyoruz.*/
            Console.WriteLine(tipimiz.Name.ToString()+" sınıfındaki metod sayısı:"+tipMetodlari.Length.ToString()); /* Metod
sayısını Length özelliği ile alıyoruz.*/
            /* Döngümüzü oluşturuyor ve Metodlarımızı bir takım özellikleri ile birlikte yazdırıyoruz.*/
            for(int i=0;i<tipMetodlari.Length;++i)
            {
                Console.WriteLine("Metod adi:"+tipMetodlari[i].Name.ToString()+" |Dönüş
değeri:"+tipMetodlari[i].ReturnType.ToString()); /* Metodun ismini name özelliği ile alıyoruz. Metodun dönüş tipini ReturnType
özelliği ile alıyoruz. */
                System.Reflection.ParameterInfo[] prmInfo=tipMetodlari[i].GetParameters();
                /* Bu satırda ise, i indeksli metoda ait parametre bilgilerini GetParameters metodu ile alıyor ve Reflection uzayında
bulunan ParameterInfo sınıfı tipinden bir diziye aktarıyoruz. Böylece ilgili metodun parametrelerine ve parametre bilgilerine
erişebiliriz.*/
                Console.WriteLine("-----Parametre Bilgileri-----"+prmInfo.Length.ToString()+" parametre");
                /* Döngümüz ile i indeksli metoda ait parametrelerin isimlerini ve tiplerini yazdırıyoruz. Bunun için Name ve
ParameterType metodlarını kullanıyoruz.*/
                for(int j=0;j<prmInfo.Length;++j)
                {
                    Console.WriteLine("P.Adi:"+prmInfo[j].Name.ToString()+" |P.Tipi:"+prmInfo[j].ParameterType.ToString());
                }
                Console.WriteLine("-----");
            }
        }
    }
}

```

Şimdi uygulamamızı çalıştıralım ve sonuçlarına bakalım.

```

C:\ "D:\vs samples\ReflectionSample2\bin\Debug\ReflectionSample2.exe"

Metod adi:Select !Dönüs degeri:System.Data.DataRow[]
-----Parametre Bilgileri-----2 parametre
P.Adi:filterExpression !P.Tipi:System.String
P.Adi:sort !P.Tipi:System.String

Metod adi:Select !Dönüs degeri:System.Data.DataRow[]
-----Parametre Bilgileri-----3 parametre
P.Adi:filterExpression !P.Tipi:System.String
P.Adi:sort !P.Tipi:System.String
P.Adi:recordStates !P.Tipi:System.Data.DataViewRowState
-----
Metod adi:BeginLoadData !Dönüs degeri:System.Void
-----Parametre Bilgileri-----0 parametre
-----
Metod adi:EndLoadData !Dönüs degeri:System.Void
-----Parametre Bilgileri-----0 parametre
-----
Metod adi:LoadDataRow !Dönüs degeri:System.Data.DataRow
-----Parametre Bilgileri-----2 parametre
P.Adi:values !P.Tipi:System.Object[]
P.Adi:fAcceptChanges !P.Tipi:System.Boolean
-----
Metod adi:GetType !Dönüs degeri:System.Type
-----Parametre Bilgileri-----0 parametre

```

Örneğin 3 parametrelili
Select metodu

Şekil 2. System.Data.DataTable tipinin metodları ve metod parametrelerine ait bilgiler.

Reflection teknikleri yardımıyla çalıştırdığımız programa ait sınıflarında bilgilerini elde edebiliriz. İzleyen örneğimizde, yazdığımız bir sınıfa ait üye bilgilerine bakacağız. Üyelerine bakacağımız sınıfın kodları;

```

using System;
namespace ReflectionSample3
{
    public class OrnekSinif
    {
        private int deger;
        public int Deger
        {
            get
            {
                return deger;
            }
            set
            {
                deger=value;
            }
        }
        public string metod(string a)
        {

```



```

        return "Burak Selim SENYURT";
    }
    int yas=27;
    string dogum="istanbul";
}
}

```

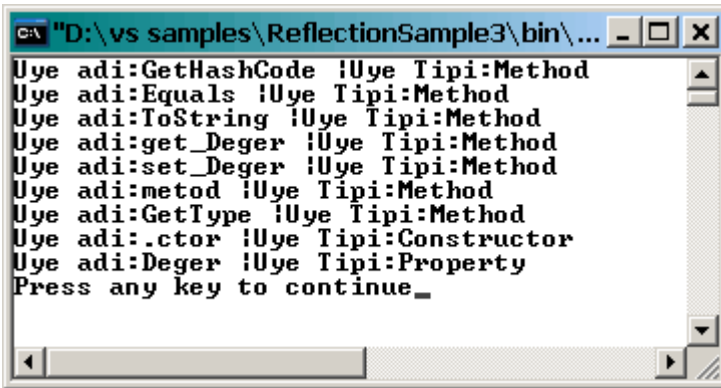
ikinci sınıfımızın kodları;

```

using System;
using System.Reflection;
namespace ReflectionSample3
{
    class Class1
    {
        static void Main(string[] args)
        {
            Type tipimiz=Type.GetType("ReflectionSample3.OrnekSinif");
            MemberInfo[] tipUyeleri=tipimiz.GetMembers();
            for(int i=0;i<tipUyeleri.Length;++i)
            {
                Console.WriteLine("Uye adi:"+tipUyeleri[i].Name.ToString()+" |Uye Tipi:"+tipUyeleri[i].MemberType.ToString());
            }
        }
    }
}

```

Şimdi uygulamamızı çalıştıralım.



Şekil 3. Kendi yazdığımı sınıf üyelerinede bakabiliriz.

Peki kendi sınıfımıza ait bilgileri edinmenin bize ne gibi bir faydası olabilir. İşte şimdi tam dışımıza göre bir örnek yazacağız. Örneğimizde, bir tablodaki verileri bir sınıf içersinden tanımladığımız özelliklere alacağız. Bu uygulamamız sayesinde sadece tek satırlık bir kod ile, herhangi bir kontrolü veriler ile doldurabileceğiz. Bu uygulamada esas olarak, veriler veritabanındaki tablodan alınacak ve oluşturduğumuz bir koleksiyon sınıfından bir diziye aktarılacak. Oluşturulan bu koleksiyon dizisi, bir DataGridView kontrolü ile ilişkilendirilecek. Teknik olarak kodumuzda, Reflection uzayının PropertyInfo sınıfını kullanarak, oluşturduğumuz sınıfa ait özellikler ile tablodaki alanları karşılaştıracak ve uygun iseler bu özelliklere tabloda karşılık gelen alanlar içindeki değerleri aktaracağız. Dilerseniz kodumuz yazmaya başlayalım.

```
using System;
using System.Reflection;
using System.Data;
using System.Data.SqlClient;
namespace ReflectDoldur
{
    /* Kitap sınıfı KitapKoleksiyonu isimli koleksiyon içinde saklayacağımız değerlerin tipi olacaktır ve iki adet özellik içerecektir.
    Bunlardan birisi, tablomuzdaki Adi alanının değerini, ikincisi ise BasimEvi alanının değerini tutacaktır.*/
    public class Kitap
    {
        private string kitapAdi;
        private string yayimci;
        /* Özelliklerimizin adlarının tablomuzdan alacağımız alan adları ile aynı olmasına dikkat edelim.*/
        public string Adi
        {
            get
            {
                return kitapAdi;
            }
            set
            {
                kitapAdi=value;
            }
        }
        public string BasimEvi
        {
            get
            {
                return yayimci;
            }
            set
```

```

    {
        yayimci=value;
    }
}

/* Yapıcı metodumuz parametre olarak geçirilen bir DataRow değişkenine sahip. Bu değişken ile o anki satırı alıyoruz.*/
public Kitap(System.Data.DataRow dr)
{
    PropertyInfo[] propInfos=this.GetType().GetProperties(); /* this ile bu sınıfı temsil ediyoruz. Bu sınıfın tipini alıp bu tipe
ait özellikleri elde ediyor ve bunları PropertyInfo sınıfı tipinden diziye aktarıyoruz.*/

    /* Döngümüz ile tüm özellikleri geziyoruz. Eğer metodumuza parametre olarak geçirilen dataRow değişkenimiz, bu
özelliğin adında bir alan içeriyorsa, bu özelliğe ait SetValue metodunu kullanarak özelliğimize, ilgili tablo alanının değerini
aktarıyoruz.*/
    for(int i=0;i<propInfos.Length;++i)
    {
        if(propInfos[i].CanWrite) /* Burada özelliğimizin bir Set bloğuna sahip olup olmadığına bakılıyor. Yani özelliğimizin
yazılabilir olup olmadığına. Bunu sağlayan özelliğimiz CanWrite. Eğer özellik yazılabilir ise (yada başka bir deyişle readonly değil ise)
true değerini döndürür.*/
        {
            try
            {
                if(dr[propInfos[i].Name]!=null) /* dataRow değişkeninde, özelliğimin adındaki alanın değerine bakılıyor. Null
değer değil ise SetValue ile alanın değeri özelliğimize yazdırılıyor. */
                {
                    propInfos[i].SetValue(this,dr[propInfos[i].Name],null);
                }
            }
            else
            {
                propInfos[i].SetValue(this,null,null);
            }
        }
        catch
        {
            propInfos[i].SetValue(this,null,null);
        }
    }
}
}

/* KitapKoleksiyonu sınıfımız bir koleksiyon olacak ve tablomuzdan alacağımız iki alana ait verileri Kitap isimli nesnelerde

```

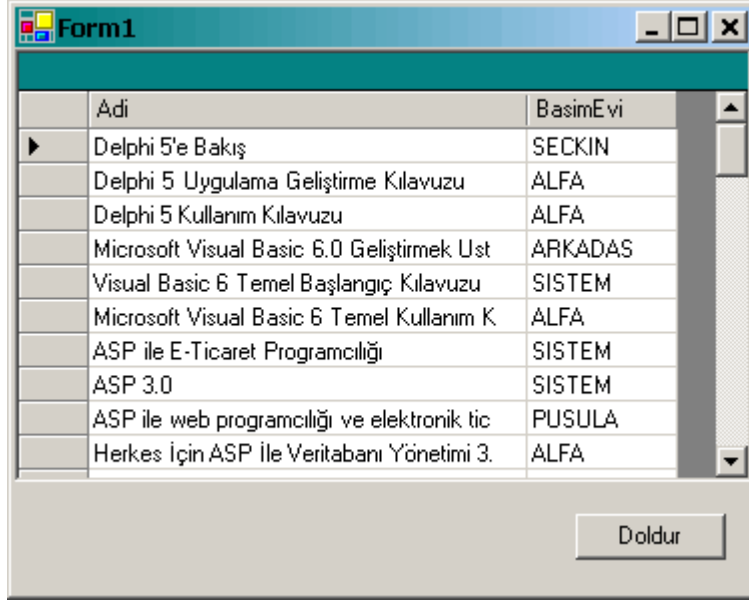
saklanması sağlayacak. Bu nedenle, bir CollectionBase sınıfından türetildi. Böylece sınıfımız içinde indeksleyiciler kullanabileceğiz.*/

```
public class KitapKoleksiyonu: System.Collections.CollectionBase
{
    public KitapKoleksiyonu()
    {
        SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
        SqlDataAdapter da=new SqlDataAdapter("Select Adi,BasimEvi From Kitaplar",conFriends);
        DataTable dtKitap=new DataTable();
        da.Fill(dtKitap);
        foreach(DataRow drow in dtKitap.Rows)
        {
            this.InnerList.Add(new Kitap(drow));
        }
    }
    public virtual void Add(Kitap _kitap)
    {
        this.List.Add(_kitap);
    }
    public virtual Kitap this[int Index]
    {
        get
        {
            return (Kitap)this.List[Index];
        }
    }
}
}
```

Ve işte formumuzda kullandığımız tek satırlık kod;

```
private void btnDoldur_Click(object sender, System.EventArgs e)
{
    dataGrid1.DataSource = new ReflectDoldur.KitapKoleksiyonu();
}
```

Şimdi uygulamamızı çalıştıralım ve bakalım.



Şekil 4. Programın Çalışmasının Sonucu

Görüldüğü gibi tablomuzdaki iki Alana ait veriler yazdığımız KitapKoleksiyonu sınıfı yardımıyla, her biri Kitap tipinde bir nesne alan koleksiyonumuza eklenmiş ve bu sonuçlarda dataGrid kontrolümüze bağlanmıştır. Siz bu örneği dahada iyi bir şekilde geliştirebilirsiniz. Umuyorumki bu örnekte yapmak istediğimizi anlamışsınızdır. Yansıma tekniğini bu kod içinde kısa bir yerde kullandık. Sınıfın özelliklerinin isminin, tablodaki alanların ismi ile aynı olup olmadığını ve aynı isler yazılabilir olup olmadıklarını öğrenmekte kullandık. Değerli Okurlarım. Geldik bir makalemizin daha sonuna. Hepinize mutlu günler dilerim.

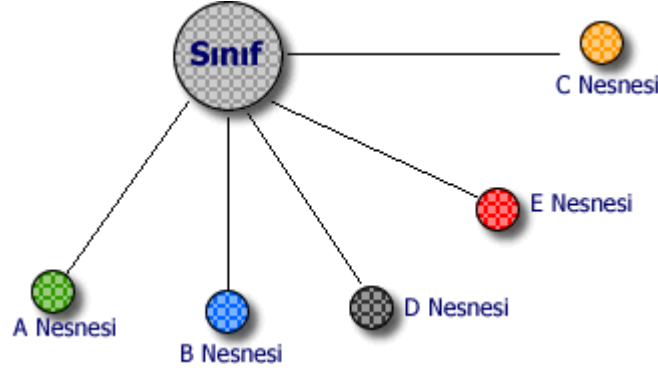
Bir Sınıf Yazalım

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde ADO.NET kavramı içerisinde sınıfları nasıl kullanabileceğimizi incelemeye çalışacak ve sınıf kavramına kısa bir giriş yapacağız. Nitekim C# dili tam anlamıyla nesne yönelimli bir dildir. Bu dil içerisinde sınıf kavramının önemli bir yeri vardır. Bu kavramı iyi anlamak, her türlü teknikte, sınıfların avantajlarından yararlanmanızı ve kendinize özgü nesnelere sahip olabilmenizi sağlar. Zaten .net teknolojisinde yer alan her nesne, mutlaka sınıflardan türetilmektedir.

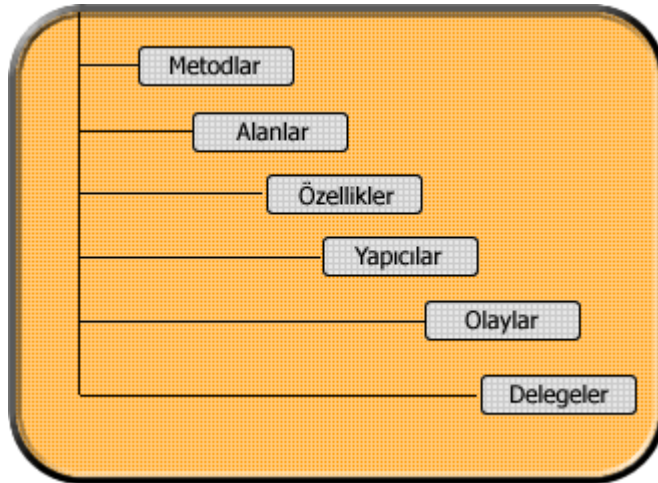
Çevremize baktığımız zaman, çok çeşitli canlılar görürüz. Örneğin çiçekler. Dünya üzerinde kaç tür(cins) çiçek olduğunu bilmeniz var mı ? Ama biz bir çiçek gördüğümüzde ona çoğunlukla "Çiçek" diye hitap ederiz özellikle adını bilmiyorsak. Sonra ise bu çiçeğin renginden, yapraklarının şeklinden, ait olduğu türden, adından bahsederiz. Çiçek tüm bu çiçekler için temel bir sınıf olarak kabul edilebilir. Dünya üzerindeki tüm çiçekler için ortak nitelikleri vardır. Her çiçeğin bir renginin(renklerinin) olması gibi. İşte nesne yönelimli programlama kavramında bahsedilen ve her şeyin temelini oluşturan sınıf kavramı bu benzetme ile tamamen aynıdır. Çiçek bir sınıf olarak algılanırken, sokakta gördüğümüz her çiçek bu sınıfın ortak özelliklerine sahip birer nesne olarak nitelendirilebilir. Ancak tabiki çiçekler arasında da türler mevcuttur. Bu türler ise, Çiçek temel sınıfından türeyen kendi belirli özellikleri dışında Çiçek sınıfının özelliklerindeki kalıtsal olarak alan başka sınıflardır. Bu yaklaşım inheritance (kalıtım) kavramı olarak ele alınır ve nesne yönelimli programlamanın temel üç ögesinden biridir. Kalıtım konusuna ve diğerlerine ilerliyen makalelerimizde değinmeye çalışacağız.

Bugün yapacağımız bir sınıfın temel yapı taşlarına kısaca değinmek ve kendimize ait işimize yarayabilecek bir sınıf tasarlamak. Çiçek sınıfından gerçek C# ortamına geçtiğimizde, her şeyin bir nesne olduğunu görürüz. Ancak her nesne temel olarak Object sınıfından türemektedir. Yani herşeyin üstünde bir sınıf kavramı vardır. Sınıflar, bir takım üyelere sahiptir. Bu üyeler, bu sınıftan örnekendirilen nesneler için farklı değerlere sahip olurlar. Yani bir sınıf varken, bu sınıftan örnekendirilmiş n sayıda nesne oluşturabiliriz. Kaldıki bu nesnelerin her biri, tanımlandığı sınıf için ayrı ayrı özelliklere sahip olabilirler.



Şekil 1. Sınıf (Class) ve Nesne (Object) Kavramı

Bir sınıf kendisinden oluşturulacak nesneler için bir takım üyeler içermelidir. Bu üyeler, alanlar (fields), metodlar (methods), yapıcılar (constructor), özellikler (properties), olaylar(events), delegeler (delegates) vb... dır. Alanlar verileri sınıf içerisinde tutmak amacıyla kullanılırlar. Bir takım işlevleri veya fonksiyonellikleri gerçekleştirmek için metodları kullanırız. Çoğunlukla sınıf içinde yer alan alanların veya özelliklerin ilk değerlerin atanması gibi hazırlık işlemlerinde ise yapıcılar kullanırız. Özellikler kapsülleme dediğimiz Encapsulating kavramının bir parçasıdır. Çoğunlukla, sınıf içersinden tanımladığımız alanlara, dışarıdan doğrudan erişilmesini istemeyiz. Bunun yerine bu alanlara erişen özellikleri kullanırız. İşte bu sınıf içindeki verileri dış dünyadan soyutlamaktır yani kapsüllemektir. Bir sınıfın genel hatları ile içereceği üyeleri aşağıdaki şekilde de görebilirsiniz.



Şekil 2 . Bir sınıfın üyeleri.

Sınıflar ile ilgili bu kısa bilgilerden sonra dilerseniz sınıf kavramını daha iyi anlamamızı sağlayacak basit bir örnek geliştirelim. Sınıflar ve üyeleri ile ilgili diğer kavramları kodlar içerisinde yer alan yorum satırlarında açıklamaya devam edeceğiz. Bu örnek çalışmamızda, Sql Suncusuna bağlanırken, bağlantı işlemlerini kolaylaştıracak birtakım üyeler sağlayan bir sınıf geliştirmeye çalışacağız. Kodları yazdıkça bunu çok daha iyi anlayacaksınız. İşte bu uygulama için geliştirdiğimiz, veri isimli sınıfımızın kodları.

```

using System;
using System.Data.SqlClient;
namespace Veriler /* Sınıfımız Veriler isimli isim uzayında yer alıyor. Çoğu zaman aynı isme sahip sınıflara sahip olabiliriz. İşte bu
gibi durumlarda isim uzayları bu sınıfların birbirinden farklı olduğunu anlamamıza yardımcı olurlar.*/
{
    public class Veri /* Sınıfımızın adı Veri */
    {
        /* İzleyen satırlarda alan tanımlamalarının yapıldığını görmekteyiz. Bu alanlar private olarak tanımlanmıştır. Yani sadece bu
sınıf içerisinde erişilebilir ve değerleri değiştirilebilir. Bu alanları tanımladığımız özelliklerin değerlerini tutmak amacıyla
tanımlıyoruz. Amacımız bu değerlere sınıf dışından doğrudan erişilmesini engellemek.*/
        private string SunucuAdi;
        private string VeritabaniAdi;
        private string Kullanici;
        private string Parola;
        private SqlConnection Kon; /* Burada SqlConnection tipinden bir değişken tanımladık. */
        private bool BaglantiDurumu; /* Sql sunucumuza olan bağlantının açık olup olmadığına bakacağız.*/
        private string HataDurumu; /* Sql sunucusuna bağlanırken hata olup olmadığına bakacağız.*/
        /* Aşağıda sunucu adında bir özellik tanımladık. Herbir özellik, get veya set bloklarından en az birini içermek zorundadır. */
        public string sunucu /* public tipteki üyelere sınıf içinden, sınıf dışından veya türetilmiş sınıflardan yani kısaca her yerden
erişilebilmektedir.*/
        {
            get
            {
                return SunucuAdi; /* Get ile, sunucu isimli özelliğe bu sınıfın bir örneğinden erişildiğinde okunacak değerin
alınabilmesi sağlanır . Bu değer bizim private olarak tanımladığımız SunucuAdi değişkeninin değeridir. */
            }
            set
            {
                SunucuAdi=value; /* Set bloğunda ise, bu özelliğe, bu sınıfın bir örneğinden değer atamak istediğimizde yani
özellik gösterdiği private SunucuAdi alanının değerini değiştirmek için kullanırız. Özelliğe sınıf örneğinden atanan değer, value
olarak taşınmakta ve SunucuAdi alanına aktarılmaktadır.*/
            }
        }
        public string veritabani
        {
            get
            {
                return VeritabaniAdi;
            }
        }
    }
}

```

```

    set
    {
        VeritabaniAdi=value;
    }
}

```

public string kullanıcı /* Bu özellik sadece set bloğuna sahip olduğu için sadece değer atanabilir ama içeriği görüntülenemez.

Yani kullanıcı özelliğini bir sınıf örneğinde, Kullanıcı private alanının değerini öğrenmek için kullanamayız.*/

```

{
    set
    {
        Kullanici=value;
    }
}

```

public string parola

```

{
    set
    {
        Parola=value;
    }
}

```

public SqlConnection con /* Buradaki özellik SqlConnection nesne türündendir ve sadece okunabilir bir özelliktir. Nitekim sadece get bloğuna sahiptir. */

```

{
    get
    {
        return Kon;
    }
}

```

public bool baglantiDurumu

```

{
    get
    {
        return BaglantiDurumu;
    }
}

```

set /* Burada set bloğunda başka kodlar da ekledik. Kullanıcımız bu sınıf örneği ile bir Sql bağlantısı yarattıktan sonra eğer bu bağlantıyı açmak isterse baglantiDurumu özelliğine true değerini göndermesi yeterli olacaktır. Eğer false değeri gönderirse bağlantı kapatılır. Bu işlemleri gerçekleştirmek için ise BaglantiAc ve BaglantiKapat isimli sadece bu sınıfa özel olan private metodlarımızı kullanıyoruz.*/

```

{

```



```

        BaglantiDurumu=value;
        if(value==true)
        {
            BaglantiAc();
        }
        else
        {
            BaglantiKapat();
        }
    }
}

public string hataDurumu
{
    get
    {
        return HataDurumu;
    }
}

```

public Veri() /* Her sınıf mutlaka hiç bir parametresi olmayan ve yandaki satırda görüldüğü gibi, sınıf adı ile aynı isme sahip bir metod içerir. Bu metod sınıfın yapıcı metodudur. Yani Constructor metodudur. Bir yapıcı metod içerisinde çoğunlukla, sınıf içinde kullanılan alanlara başlangıç değerleri atanır veya ilk atamalar yapılır. Eğer siz bir yapıcı metod tanımlamaz iseniz, derleyici aynen bu metod gibi boş bir yapıcı oluşturacak ve sayısal alanlara 0, mantıksal alanlara false ve string alanlara null başlangıç değerlerini atayacaktır.*/

```

{
}

```

/* Burada biz bu sınıfın yapıcı metodunu aşırı yüklüyoruz. Bu sınıftan bir nesneyi izleyen yapılandırıcı ile oluşturabiliriz. Bu durumda yapıcı metod içerdiği dört parametreyi alacaktır. Metodun amacı ise belirtilen değerlere göre bir Sql bağlantısı yaratmaktır.*/

```

public Veri(string sunucuAdi,string veritabaniAdi,string kullanıcıAdi,string sifre)
{
    SunucuAdi=sunucuAdi;
    VeritabaniAdi=veritabaniAdi;
    Kullanici=kullanıcıAdi;
    Parola=sifre;
    Baglan();
}

```

/* Burada bir metod tanımladık. Bu metod ile bir Sql bağlantısı oluşturuyoruz. Eğer bir metod geriye herhangi bir değer göndermeyecek ise yani vb.net teki fonksiyonlar gibi çalışmayacak ise void olarak tanımlanır. Ayrıca metodumuzun sadece bu sınıf içerisinde kullanılmasını istediğimiz için private olarak tanımladık. Bu sayede bu sınıf dışından örneğin formumuzdan

ulaşamamalarını sağlamış oluyoruz.*/

```

private void Baglan()
{
    SqlConnection con=new SqlConnection("data source="+SunucuAdi+";initial catalog="+VeritabaniAdi+";user
id="+Kullanici+";password="+Parola);
    Kon=con;
}
/* Bu metod ile Sql sunucumuza olan bağlantıyı açıyoruz ve BaglantiDurumu alanına true değerini aktarıyoruz.*/
private void BaglantiAc() /* Bu metod private tanımlanmıştır. Çünkü sadece bu sınıf içerisinde çağırılabilir istiyoruz. */
{
    Kon.Open();
    try
    {
        BaglantiDurumu=true;
        HataDurumu="Baglanti sağlandı";
    }
    catch(Exception h)
    {
        HataDurumu="Baglanti Sağlanamadı. "+h.Message.ToString();
    }
}
/* Bu metod ilede Sql bağlantımızı kapatıyor ve BaglantiDurumu isimli alanımıza false değerini aktarıyoruz.*/
private void BaglantiKapat()
{
    Kon.Close();
    BaglantiDurumu=false;
}
}

```

Şimdi ise sınıfımızı kullandığımız örnek uygulama formunu tasarlayalım . Bu uygulamamız aşağıdaki form ve kontrollerinden oluşuyor.

Şekil 3. Form Tasarımımız.

Formumuza ait kodlar ise şöyle.

Veriler.Verı v;

```
private void btnBaglan_Click(object sender, System.EventArgs e)
```

```
{
```

```
    /* Bir sınıf örneği yaratmak için new anahtar kelimesini kullanırız. New anahtar kelimesi bize kullanabileceğimiz tüm yapıcı
    metodları gösterecektir. (IntelliSense özelliği). */
```

```
    v=new Veri(txtSunucu.Text,txtVeritabanı.Text,txtKullanıcı.Text,txtParola.Text);
```

```
Veriler.Verı v=new Veri{
```

```
    ▲ 2 of 2 ▼ Veri.Verı (string sunucuAdı, string veritabanıAdı, string kullanıcıAdı, string sifre)
```

```
}
```

```
private void btnAc_Click(object sender, System.EventArgs e)
```

```
{
```

```
    v.baglantiDurumu=true;
```

```
    stbDurum.Text="Sunucu Bağlantısı Açık? "+v.baglantiDurumu.ToString();
```

```
}
```

```
private void btnKapat_Click(object sender, System.EventArgs e)
```

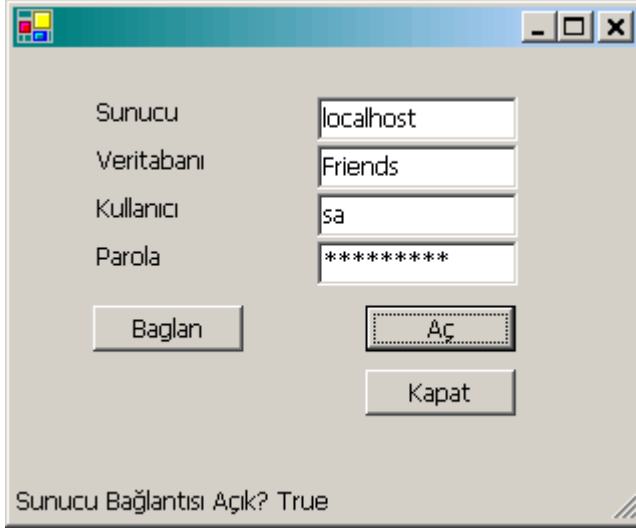
```
{
```

```
    v.baglantiDurumu=false;
```

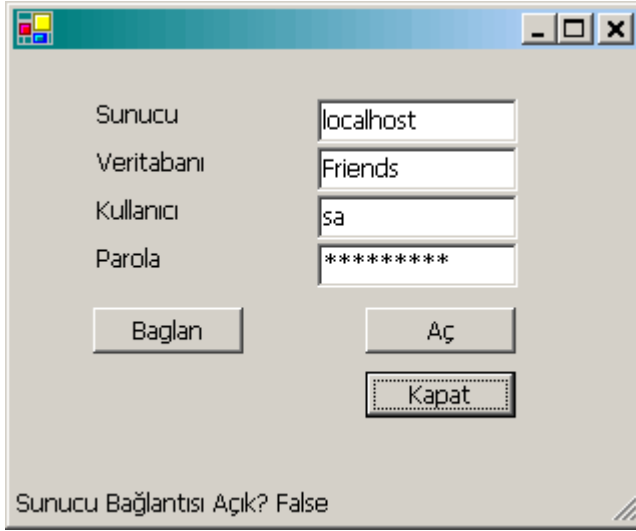
```
    stbDurum.Text="Sunucu Bağlantısı Açık? "+v.baglantiDurumu.ToString();
```

```
}
```

Şimdi uygulamamızı bir çalıştıralım.



Şekil 5. Bağlantıyı açmamız halinde.



Şekil 6. Bağlantıyı kapatmamız halinde.

Değerli okurlarım, ben bu sınıfın geliştirilmesini size bırakıyorum. Umarım sınıf kavramı ile ilgili bilgilerimizi hatırlamış ve yeni ufuklara yelken açmaya hazır hale gelmişsinizdir. Bir sonraki makalemizde sınıflar arasında kalıtım kavramına bakacak ve böylece nesneye dayalı programlama terminolojisinin en önemli kavramlarından birini incelemeye çalışacağız. Hepinize mutlu günler dilerim.

Virtual(Sanal) Metodlar

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde sanal metodların kalıtım içerisindeki rolüne bakacağız. Sanal metodlar, temel sınıflarda tanımlanan ve türeyen sınıflarda geçersiz kılınabilen metodlardır. Bu tanım bize pek bir şey ifade etmez aslında. O halde gelin sanal metodların neden kullanırız, önce buna bakalım. Bu amaçla minik bir örnek ile işe başlıyoruz.

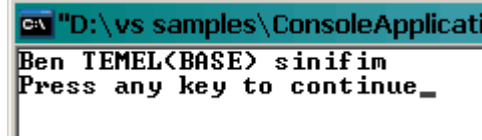
```

using System;
namespace ConsoleApplication1
{
    public class Temel
    {
        public Temel()
        {
        }
        public void Yazdir()
        {
            Console.WriteLine("Ben TEMEL(BASE) sinifim");
        }
    }
    public class Tureyen:Temel
    {
        public Tureyen()
        {
        }
        public void Yazdir()
        {
            Console.WriteLine("Ben TUREYEN(DERIVED) sinifim");
        }
    }
    class Class1
    {
        static void Main(string[] args)
        {
            Temel bs;
            Tureyen drv=new Tureyen();
            bs=drv;
            bs.Yazdir();
        }
    }
}

```

Bu örneği çalıştırmadan önce satırlarımızı bir inceleyelim. Kodumuz Temel isimli bir base class ve Tureyen isimli bir Derived Class vardır. Her iki sınıf içinde Yazdir isimli iki metod tanımlanmıştır. Main metodu içinde Temel sınıftan türettiğimiz bir nesneye (bs nesnesi) Tureyen sınıf tipinden bir nesneyi (drv nesnesi) aktarıyoruz. Ardından bs nesnemizin Yazdir isimli metodunu çağırıyoruz. Sizce derleyici hangi sınıfın yazdır metodunu çağıracaktır. Drv nesnemiz Tureyen sınıf nesnesi olduğundan ve Temel sınıftan kalıtımsal olarak türetildiğinden bs isimli nesnemize aktarılabilmiştir. Şu durumda bs isimli Temel sınıf nesnemiz drv isimli Tureyen

sınıf nesnemizi taşımaktadır. Bu tip bir atamam böyle base-derived ilişkide sınıflar için geçerli bir atamadır. Sorun bs isimli nesne için Yazdir metodunun çağırılmasıdır. Biz burada Tureyen sınıf nesnesini aktardığımız için bu sınıfa ait Yazdir metodunun çalıştırılmasını bekleriz. Oysaki sonuç aşağıdaki gibi olacaktır.



```
C:\>"D:\vs samples\ConsoleApplicati
Ben TEMEL<BASE> sinifim
Press any key to continue_
```

Şekil 1. Temel Sınıfın Yazdir metodu çağırıldı.

Görüldüğü gibi Temel sınıfa ait Yazdir metodu çalıştırılmıştır. Bir çözüm olarak daha önceki kalıtım kavramını anlattığımız makalemizde incelediğimiz new anahtar kelimesini Tureyen isimli derived class içinde kullanmayı düşünebilirsiniz. Birde böyle deneyelim, bakalım neler olacak.

```
using System;
namespace ConsoleApplication1
{
    public class Temel
    {
        public Temel()
        {
        }
        public void Yazdir()
        {
            Console.WriteLine("Ben TEMEL(BASE) sinifim");
        }
    }
    public class Tureyen:Temel
    {
        public Tureyen()
        {
        }
        public new void Yazdir()
        {
            Console.WriteLine("Ben TUREYEN(DERIVED) sinifim");
        }
    }
    class Class1
    {
        static void Main(string[] args)
        {

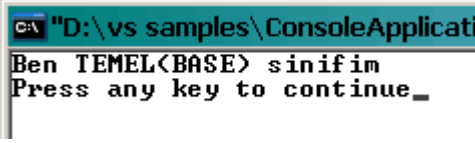
```

```

        Temel bs;
        Tureyen drv=new Tureyen();
        bs=drv;
        bs.Yazdir();
    }
}

```

Ancak new anahtar kelimesini kullanmış olsakta sonuç yine aynı olacaktır ve aşağıdaki görüntüyü alacağız.



Şekil 2. Yine `style='mso-spacerun:yes'>` Temel Sınıfın Yazdır metodu çağırıldı.

İşte bu noktada çözüm Temel sınıftaki metodumuzu Virtual(sanal) tanımlamak ve aynı metodu, Tureyen sınıf içerisinde Override (Geçersiz Kılmak) etmektir. Sanal metodların kullanım amacı budur; Base sınıfta yer alan metod yerine base sınıfa aktarılan nesnenin üretildiği derived class'taki metodu çağırmaktır.

Şimdi örneğimizi buna göre değiştirelim.

```

using System;
namespace ConsoleApplication1
{
    public class Temel
    {
        public Temel()
        {
        }
        public virtual void Yazdir()
        {
            Console.WriteLine("Ben TEMEL(BASE) sinifim");
        }
    }
    public class Tureyen:Temel
    {
        public Tureyen()
        {
        }
        public override void Yazdir()
    }
}

```

```

    {
        Console.WriteLine("Ben TUREYEN(DERIVED) sinifim");
    }
}
class Class1
{
    static void Main(string[] args)
    {
        Temel bs;
        Tureyen drv=new Tureyen();
        bs=drv;
        bs.Yazdir();
    }
}
}

```



C:\> "D:\vs samples\ConsoleApplicati
Ben TUREYEN(DERIVED) sinifim
Press any key to continue

Şekil 3. Tureyen sınıfındaki Yazdir metodu çağırılmıştır.

Burada önemli olan nokta, Temel sınıfındaki metodun virtual anahtar kelimesi ile tanımlanması, Tureyen sınıfındaki metodun ise override anahtar kelimesi ile tanımlanmış olmasıdır. Sanal metodları kullanırken dikkat etmemiz gereken bir takım noktalar vardır. Bu noktalar;

1	İki metodda aynı isime sahip olmalıdır.
2	İki metodda aynı tür ve sayıda parametre almalıdır.
3	İki metodunda geri dönüş değerleri aynı olmalıdır.
4	Metodların erişim haklarını aynı olmalıdır. Biri public tanımlanmış ise diğeri de public olmalıdır.
5	Temel sınıftaki metodu türeyen sınıfta override (geçersiz) hale getimez ise metod geçersiz kılınmaz.
6	Sadece virtual olarak tanımlanmış metodları override edebiliriz. Herhangibir base class yöntemini türeyen sınıfta override edemeyiz.

Bu noktalara dikkat etmemiz gerekmektedir. Değerli Okurlarım, geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

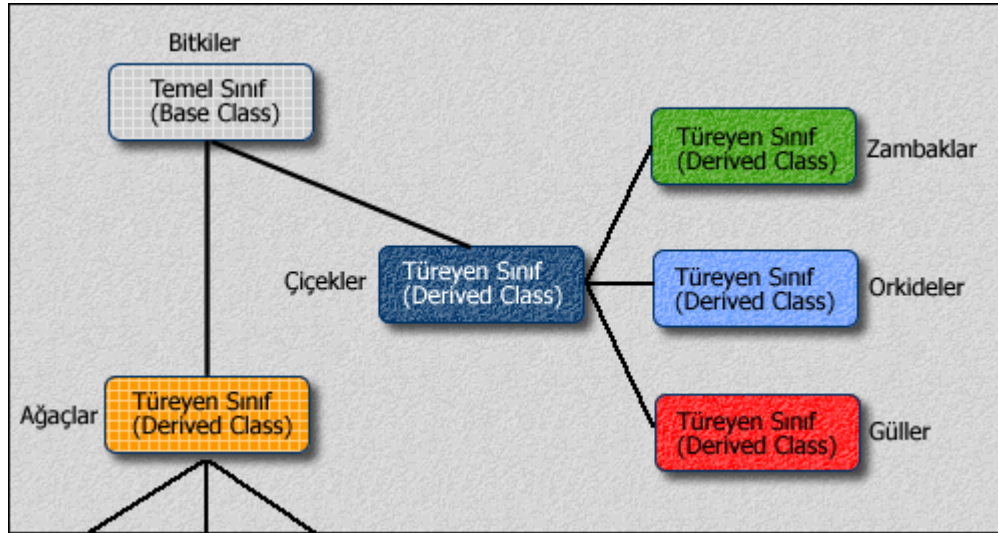
Kalıtım (Inheritance) Kavramına Kısa Bir Bakış

Değerli Okurlarım, Merhabalar.

Bir önceki makalemizde C# dilinde sınıf kavramına bir giriş yapmış ve OOP(Objcet Oriented Programming-Nesneye Dayalı Programlama) tekniğinin en önemli kavramlarından biri olan kalıttan bahsedeceğimizi söylemiştik. Bugünkü makalemizde bu kavramı incelemeye çalışacağız.

Kalıtım kavramı için verilebilecek en güzel örnekler doğamızda yer almaktadır. Örneğin Bitkiler, türlerine göre sınıflandırılırlar. Ancak hepsinin birer bitki olması ortak bir takım özelliklere sahip oldukları anlamında gelmektedir. Bitki isimli bir sınıfı göz önüne alıp Ağaçlar,Çiçekler,Deniz Bitkileri vb... gibi alt sınıflara ayırabiliriz. Tüm bu alt sınıflar Bitki sınıfından gelmekte, yani türemektedirler. Aynı şekilde Bitki türü olan bir sınıf kendi içinde başka alt sınıflara ayrılabilir. Örneğin, Çiçek sınıfı Güller, Orkideler, Papatyalar vb... Tüm bu nesneler hiyerarşik bir yapıda ele alındıklarında temel bir sınıftan türedikleri görülebilmektedir.

Temel bir nesne ve bu nesneden bir takım özellikleri almış ve başka ek özelliklere sahip olan nesneler bir araya getirildiklerinde aralarındaki ilişkinin kalıtsal olduğundan söz ederiz. İşte nesneye dayalı programlama dillerinde en önemli kavramlarından birisi kalıttır. Ortak özelliklere sahip olan tüm nesneleriniz için bir sınıf yazarsınız. Ancak elinizde bu ortak özelliklerin yanında başka ek özelliklere sahip olacak veya ortak özelliklerden bir kaçını farklı şekillerde değerlendirecek nesnelerinizde olabilir. Bunlar için yazacağınız sınıf , ilk sınıfı temel sınıf olarak baz alacak ve bu temel sınıftan türetilcek ve kendi özellikleri ile işlevselliklerine sahip olacaktır. Konuyu daha iyi pekiştirmek amacı ile aşağıdaki şekili göz önüne alarak durumu zihninizde canlandırmaya çalışalım.



Şekil 1. Inheritance Kavramı

C# ile Temel bir sınıftan birden fazla sınıf türetebilirsiniz. Ancak bir sınıfı birden fazla sınıftan türetmeniz mümkün değildir. Bunu yapmak için arayüzler (interface) kullanılır. Türeyen bir sınıf türediği sınıfın tüm özelliklerine ve işlevlerine sahiptir ve türediği sınıftaki bu elemanların yeniden tanımlanmasına gerek yoktur. Ayrıca siz yeni özellikler ve işlevsellikler katabilirsiniz. Bu makalemizdeki örnek uygulamamızda kalıttan önemli role sahip base ve new anahtar kelimelerinin kullanımında göreceksiniz. Base ile temel sınıfa nasıl paramtere gönderebileceğimizi, temel sınıftaki metodları nasıl çağırabileceğimizi ve new anahtar sözcüğü

sayesinde türeyen sınıf içinde, temel sınıftaki ile aynı isme sahip metodların nasıl işlevsellik kazanacağını da inceleme fırsatı bulacaksınız. Dilerseniz bu kısa açıklamalardan sonra hemen örnek uygulamamızın kodlarına geçelim. Konu ile ilgili detaylı açıklamaları örnek içerisindeki yorum satırlarında bulabilirsiniz.

```
using System;
namespace Inheritance1
{
    class TemelSinif /* Öncelikle temel sınıfımızı yani Base Class'ımızı yazalım. */
    {
        private string SekilTipi; /* Sadece bu class içinde tanımlı bir alan tanımladık. Bu alana Türeyen sınıfımız (derived class)
        içerisinden de erişemeyiz. Eğer temel sınıfta yer alan bir alana sadece türeyen sınıftan erişebilmek ve başka sınıflardan erişilmesini
        engellemek istiyorsak, bu durumda bu alanı protected olarak tanımlarız. */

        /* Bir özellik tanımlıyoruz. Sadece get bloğu olduğu için yalnızca okunabilir bir özellik. */
        public string sekilTipi
        {
            get
            {
                return SekilTipi;
            }
        }

        public TemelSinif() /* Temel sınıfımızın varsayılan yapıcı metodu. */
        {
            SekilTipi="Kare";
        }

        public TemelSinif(string tip) /* Overload ettiğimiz yapıcı metodumuz. */
        {
            SekilTipi=tip;
        }

        public string SekilTipiYaz() /* String sonuç döndüren bir metod.*/
        {
            return "Bu Nesnemiz, "+SekilTipi.ToString()+" tipindedir";
        }
    }

    /* İşte türeyen sınıfımız. :TemelSinif yazımı ile, bu sınıfın TemelSinif'tan türetildiğini belirtiyoruz. Böylece, TureyenSinif class'ımız
    TemelSinif class'ının özellikleride bünyesinde barındırmış oluyor.*/
    class TureyenSinif:TemelSinif
    {
        private bool Alan;
        private bool Cevre;
```

```
private int Taban;
private int Yukseklik;
private int UcuncuKenar;
private bool Kare;
private bool Dikdortgen;
private bool Ucgen;
public bool alan
{
    get
    {
        return Alan;
    }
}
public bool cevre
{
    get
    {
        return Cevre;
    }
}
public int taban
{
    get
    {
        return Taban;
    }
}
public int yukseklik
{
    get
    {
        return Yukseklik;
    }
}
public int ucuncuKenar
{
    get
    {
        return UcuncuKenar;
    }
}
```

```
    }  
}  
public bool kare  
{  
    get  
    {  
        return Kare;  
    }  
    set  
    {  
        Kare=value;  
    }  
}  
public bool dikdortgen  
{  
    get  
    {  
        return Dikdortgen;  
    }  
    set  
    {  
        Dikdortgen=value;  
    }  
}  
public bool ucgen  
{  
    get  
    {  
        return Ucgen;  
    }  
    set  
    {  
        Ucgen=value;  
    }  
}  
public TureyenSinif():base() /* Burada base anahtar kelimesine dikkatinizi çekerim. Eğer bir TureyenSinif nesnesini bu yapıcı  
metod ile türetirsek, TemelSinif'taki yapıcı metod çalıştırılacaktır. */  
{  
}
```

public TureyenSinif(string tip,bool alan,bool cevre,int taban,int yukseklik,int ucuncuKenar):base(tip) /* Buradada base anahtar kelimesi kullanılmıştır. Ancak sadece tip isimli string değişkenimiz, TemelSinif'taki ilgili yapıcı metoda gönderilmiştir. Yani bu yapıcı metod kullanılarak bir TureyenSinif nesnesi oluşturduğumuzda, TemelSinif'taki bir tek string parametre alan yapıcı metod çağırılır ve daha sonra buraya dönülerek aşağıdaki kodlar çalıştırılır.*/

```
{
    Alan=alan;
    Cevre=cevre;
    Taban=taban;
    Yukseklik=yukseklik;
    UcuncuKenar=ucuncuKenar;
}
public double AlanBul() /* Tureyen sınıfımızda bir metod tanımlıyoruz. */
{
    if(Kare==true)
    {
        return Taban*Taban;
    }
    if(Dikdortgen==true)
    {
        return Taban*Yukseklik;
    }
    if(Ucgen==true)
    {
        return (Taban*Yukseklik)/2;
    }
    return 0;
}
public double CevreBul() /* Başka bir metod. */
{
    if(Kare==true)
    {
        return 4*Taban;
    }
    if(Dikdortgen==true)
    {
        return (2*Taban)+(2*Yukseklik);
    }
    if(Ucgen==true)
    {
```

```

        return Taban+Yukseklik+UcuncuKenar;
    }
    return 0;
}

```

public new string SekilTipiYaz() /* Buradaki new anahtar kelimesine dikkat edelim. SekilTipiYaz metodunun aynısı TemelSinif class'ımızda yer almaktadır. Ancak bir new anahtar kelimesi ile aynı isimde bir metodu TureyenSinif class'ımız içinde tanımlamış oluyoruz. Bu durumda, TureyenSinif class'ından oluşturulan bir nesneye ait SekilTipiYaz metodu çağırılırsa, buradaki kodlar çalıştırılır. Oysaki new anahtar kelimesini kullanmasaydık, TemelSinif içindeki SekilTipiYaz metodunun çalıştırılacağını görecektik. */

```

{
    if(Kare==true)
    {
        return "kare";
    }
    if(Dikdortgen==true)
    {
        return "dikdortgen";
    }
    if(Ucgen==true)
    {
        return "üçgen";
    }
    return "Belirsiz";
}
}

```

```
class Class1
```

```

{
    static void Main(string[] args)
    {

```

```
        /* Önce TemelSinif tipinde bir nesne oluşturup SekilTipiYaz metodunu çağırıyoruz.*/
```

```
        TemelSinif ts1=new TemelSinif();
```

```
        Console.WriteLine(ts1.SekilTipiYaz());
```

TemelSinif ts2=new TemelSinif("Dikdörtgen"); /* Bu kes TemelSinif tipinden bir nesneyi diğer yapıcı metodu ile çağırıyor ve SekilTipiYaz metodunu çalıştırıyoruz. */

```
        Console.WriteLine(ts2.SekilTipiYaz());
```

/* Şimdi ise TureyenSinif'tan bir nesne oluşturduk ve sekilTipi isimli özelliğin değerini aldık. Kodlara bakacak olursanız sekilTipi özelliğinin TemelSinif içinde tanımlandığını görürsünüz. Yani TureyenSinif nesnemizden, TemelSinif class'ındaki izin verilen alanlara,metodlara vb.. ulaşabilmekteyiz.*/

```
        TureyenSinif tur1=new TureyenSinif();
```

```
Console.WriteLine(tur1.sekilTipi);
```

/* Şimdi ise başka bir TureyenSinif nesnesi tanımladık. Yapıcı metodumuz'un ilk parametresinin değeri olan "Benim Şeklim" ifadesi base anahtar kelimesi nedeni ile, TemelSinif class'ındaki ilgili yapıcı metoda gönderilir. Diğer parametreler ise TureyenSinif class'ı içinde işlenirler. Bu durumda tur2 isimli TureyenSinif nesnemizden TemelSinifa ait sekilTipi özelliğini çağırdığımızda, ekrana Benim Şeklim yazdığını görürüz. Çünkü bu değer TemelSinif class'ımızda işlenmiş ve bu class'taki özelliğe atanmıştır. Bunu sağlayan base anahtar kelimesidir.*/

```
TureyenSinif tur2=new TureyenSinif("Benim Şeklim",true,true,2,4,0);
```

```
Console.WriteLine(tur2.sekilTipi);
```

```
tur2.dikdortgen=true;
```

Console.WriteLine(tur2.SekilTipiYaz()); /* Tureyen sınıf içinde tanımladığımız SekilTipiYaz metodu çalıştırılır. Eğer, TureyenSinif içinde bu metodu new ile tanımlamasaydık TemelSinif class'ı içinde yer alan aynı isimdeki metod çalıştırılırdı.*/

```
if(tur2.alan==true)
```

```
{
```

```
    Console.WriteLine(tur2.sekilTipi+" Alanı:"+tur2.AlanBul());
```

```
}
```

```
if(tur2.cevre==true)
```

```
{
```

```
    Console.WriteLine(tur2.sekilTipi+" Çevresi:"+tur2.CevreBul());
```

```
}
```

```
TureyenSinif tur3=new TureyenSinif("Benim üçgenim",true,false,10,10,10);
```

```
Console.WriteLine(tur3.sekilTipi);
```

```
tur3.ucgen=true;
```

```
Console.WriteLine(tur3.SekilTipiYaz());
```

```
Console.WriteLine(tur3.sekilTipi+" Alanı:"+tur3.AlanBul());
```

```
if(tur2.cevre==true)
```

```
{
```

```
    Console.WriteLine(tur3.sekilTipi+" Çevresi:"+tur3.CevreBul());
```

```
}
```

```
}
```

```
}
```

```
}
```

Uygulamamızı çalıştırdığımızda ekran görüntümüz aşağıdaki gibi olacaktır. Umuyorum ki kalıtım ile ilgili olarak yazmış olduğumuz bu örnek sizlere yeni fikirler verebilecektir. Örneğin çok işlevsel olması şu an için önemli değil. Ancak bir sınıfın başka bir sınıftan nasıl türetildiğine, özellikle base ve new anahtar kelimelerinin bize sağladığı avantajlara dikkat etmenizi isterim.

```

C:\ "D:\vs samples\Inheritance1\bin\Debug\
Bu Nesnemiz, Kare tipindedir
Bu Nesnemiz, Dikdörtgen tipindedir
Kare
Benim Seklin
dikdortgen
Benim Seklin Alanı:8
Benim Seklin Çevresi:12
Benim üçgenim
üçgen
Benim üçgenim Alanı:50
Benim üçgenim Çevresi:30
Press any key to continue

```

Şekil 2. Uygulamanın çalışmasının sonucu.

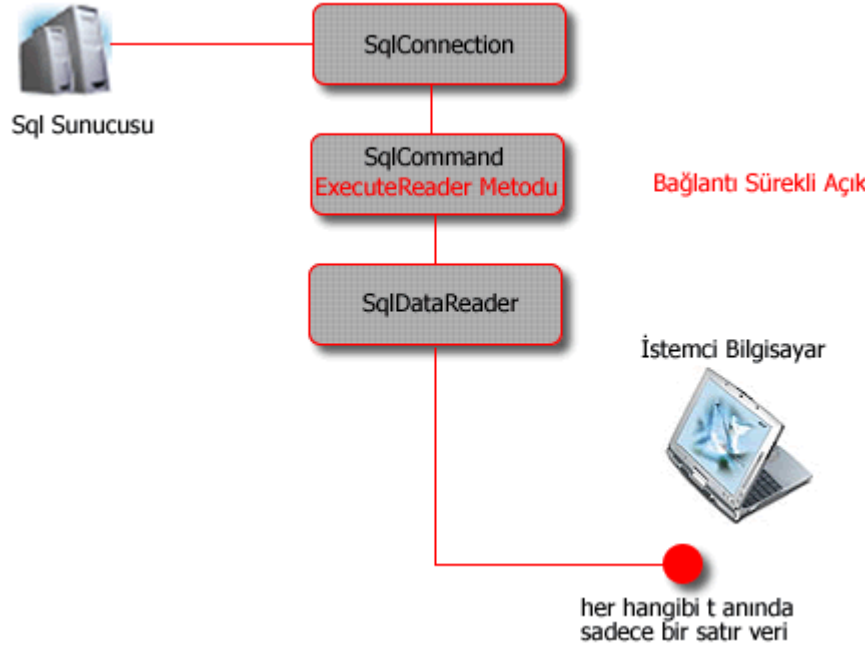
Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

SqlDataReader Sınıfı 1

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, SqlDataReader sınıfını incelemeye çalışacağız. ADO.NET'in bilgisayar programcılığına getirdiği en büyük farklıklardan birisi bağlantısız veriler ile çalışabilmemize imkan sağlamasıydı. DataSet sınıfını ve buna bağlı diğer teknikleri kastettiğimi anlamışsınızdır. Bu teknikler ile, bir veritabanı içinde yer alan tabloları, tablolar arasındaki ilişkileri, içerdikleri verileri vb... istemci makinenin belleğinde tutmamız mümkün olabiliyor. Bu sayede, bu veriler istemci makine belleği üzerinde tutulduğundan, bir veritabanına sürekli bağlantısı olan bir sisteme göre daha hızlı çalışabiliyoruz. Ayrıca veritabanı sunucusuna sürekli olarak bağlı olmadığımız için network trafiğinin hafifletmiş oluyoruz. Tüm bu hoş ayrıntılar dışında hepimiz için nafoş olan ayrıntı, her halikarda bellek tüketiminin artması. Öyleki bazen kullandığımız programlarda sadece listeleme amacı ile, sadece görsellik amacı ile küçük veriler ile çalışmak durumunda kaldığımızda, bu elde edilebilirlik için fazlasıyla kaynak tüketmiş oluyoruz. İşte ADO.NET 'te bu tip veriler düşünülerek, listeleme amacı güden, küçük boyutlarda olan veri kümeleri için DataReader sınıfları tasarlanmıştır. Biz bugün SqlDataReader sınıfını inceleyeceğiz.

Bir SqlDataReader sınıfı bir veri akımını(data stream) temsil eder. Bu nedenle herhangi bir anda bellekte sadece bir veri satırına erişebilir. İşte bu performansı ve hızı arttıran bir unsurdur. Bu veri akımını elde etmek için sürekli açık bir sunucu bağlantısı ve verileri getirecek sql sorgularını çalıştıracak bir SqlCommand nesnesi gereklidir. Dikkat edecek olursanız sürekli açık olan bir sql sunucu bağlantısından yani bir SqlConnection'dan bahsettik. SqlDataReader nesnesi sql sunucu bağlantısının sürekli açık olmasını gerektirdiği için network trafiğini meşgul eder, aynı zamanda kullandığı SqlConnection nesnesinin başka işlemler için kullanılmasında engeller. İşte bu eksiler nedeni ile, onun sadece okuma amaçlı veya listeleme amaçlı ve yalnız okunabilir veri kümeleri için kullanılması önerilir. Bu nedenle, read-only(yalnız okunabilir) ve forward-only(sadece ileri yönlü) olarak tanımlanır. Şimdi SqlDataReader nesnesi ile bir veri akımının elde edilmesi konusunu aşağıdaki şekil yardımıyla incelemeye çalışalım.

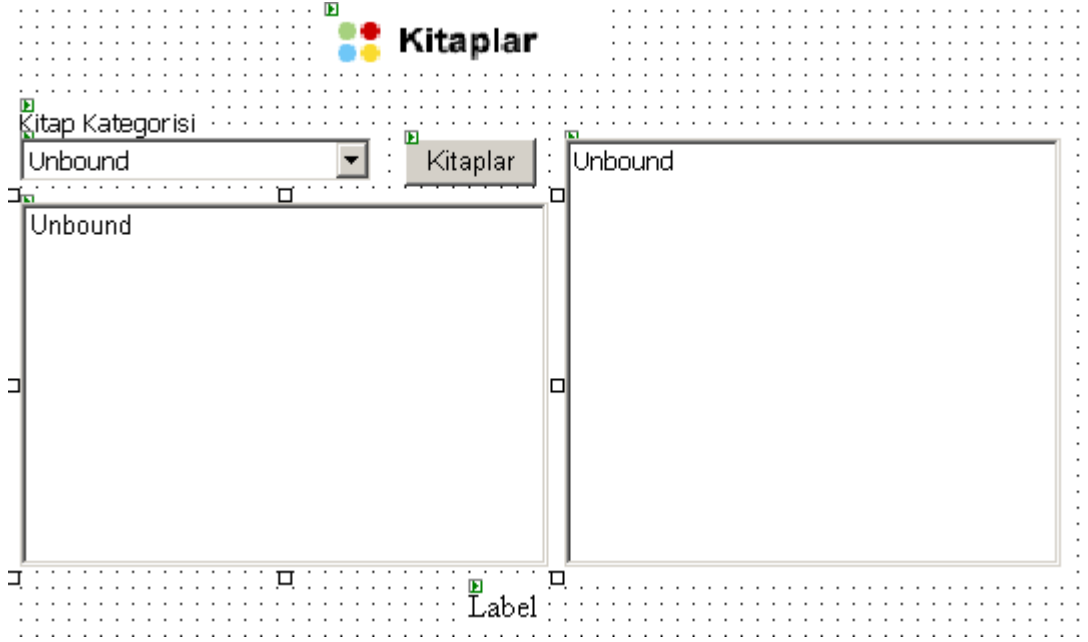


Şekil 1. SqlDataReader nesnesinin çalışma şekli.

Görüldüğü gibi sql sunucusuna sürekli bağlı durumdayız. Bunun yanında SqlCommand sorgusunun çalıştırılması sonucu elde edilen veri akımı SqlDataReader nesnesince temsil edilmektedir. İstemci bilgisayarımız, her hangibir t zamanında kendi belleğinde SqlDataReader'ın taşıdığı veri akımının sadece tek bir satırını temsil etmektedir. SqlCommand nesnesinin çalıştırdığı sql cümlecığının sonucunu(sonuç kümesini) SqlDataReader nesnemize aktarmak için ExecuteReader metodu kullanılır. Burada bir noktaya daha değinelim. SqlDataReader sınıfı herhangi yapıcı metoda sahip değildir. Yani bir SqlDataReader nesnesini bir new metodu ile oluşturamayız. Sadece bir SqlDataReader nesnesi tanımlayabiliriz. Maharet aslında SqlCommand nesnesinin ExecuteReader metodundadır. Bu metod ile, daha önceden tanımladığımız SqlDataReader nesnesini oluşturur ve SqlCommand sorgusunun sonucu olan veri kümesini temsil etmesini sağlarız. Ancak bir DataTable veya bir DataSet'te olduğu gibi sorgu sonucu elde edilen veri kümesinin tamamı bellekte saklanmaz. Bunun yerine SqlDataReader nesnesi kendisini, çalıştırılan sorgu sonuçlarını tutan geçici bir dosyanın ilk satırının öncesine konumlandırır.

İşte bu noktadan sonraki satırları okuyabilmek için bu sınıfa ait Read metodunu kullanırız. Read metodu, her zaman bir sonraki satıra konumlanılmasını sağlar. Tabi bir sonrasında kayıt olduğu sürece bu işlemi yapar. Böylece bir While döngüsü ile Read metodunu kullanırsak, sorgu sonucu elde edilen veri kümesindeki tüm satırları gezebiliriz. Konumlanılan her bir satır bellekte temsil edilir. Dolayısıyla bir sonraki t zamanında, bellekte eski satırın yerini yeni satır alır. Bu sorgu sonucu elde edilen veri kümesinden belleğe doğru devam eden bir akım(stream) dır. Geriye hareket etmek gibi bir lüksümüz olmadığı gibi, t zamanında bellekte yer alan satırları değiştirmek gibi bir imkanımızda bulunmamaktadır. İşte performansı bu arttırmaktadır. Ama elbetteki çok büyük boyutlu ve satırlı verilerle çalışırken listeleme amacımız yok ise veriler üzerinde değişiklikler yapılabilmesinide istiyorsak bu durumda bağlantısız veri elemanlarını kullanmak daha mantıklı olacaktır. Şimdi dilerseniz konumuz ile ilgili bir örnek yapalım. Özellikle ticari sitelerde, çoğunlukla kullanıcı olarak ürünleri inceleriz. Örneğin kitap alacağız. Belli bir kategorideki kitaplara bakmak istiyoruz. Kitapların sadece adları görünür olsun. Kullanıcı bir kitabı seçtiğinde, bu kitaba ait detaylarda görebilsin. Tüm bu işlemler sadece izlemek ve bakmaktan ibaret. Dolayısıyla bu veri listelerini, örneğin bir listBox kontrolüne yükleyecek isek ve sonuç olarak bir listeleme yapıcak isek, SqlDataReader nesnelerini kullanmamız daha avantajlı olacaktır. Dilerseniz uygulamamızı yazmaya

başlayalım. Öncelikle Visual Studio.Net ortamında bir Web Application oluşturalım. Default.aspx isimli sayfamızı ben aşağıdaki gibi tasarladım.



Şekil 2. WebForm tasarımıımız.

Şimdi de kodlarımızı yazalım.

```
SqlConnection conFriends;
public void baglantiAc()
{
    /* Sql Sunucumuza bağlanmak için SqlConnection nesnemizi oluşturuyoruz ve bağlantıyı açıyoruz. */
    conFriends=new SqlConnection("data source=localhost;integrated security=sspi;initial catalog=Friends");
    conFriends.Open();
}
private void Page_Load(object sender, System.EventArgs e)
{
    if(!Page.IsPostBack) /* Eğer sayfa daha önce yüklenmediyse bu if bloğu içindeki kodlar çalıştırılıyor. */
    {
        baglantiAc(); /* Sql sunucumuza olan SqlConnection nesnesini tanımlayan ve bağlantıyı açan metodumuzu çağırıyoruz. */
        /* SqlCommand nesnemize çalıştıracağımız sql sorgusunu bildiriyoruz. Bu sorgu Kitaplar tablosundan Kategori alanına ait verileri Distinct komutu sayesinde, benzersiz şekilde alır. Nitekim aynı Kategori isimleri tabloda birden fazla sayıda. Biz Distinct sayesinde birbirinden farklı kategorileri tek tek elde ediyoruz. Böylece veri kümemizizde mümkün olduğunca küçültmüş ve bir SqlDataReader nesnesinin tam dişine göre hazırlamış oluyoruz. */
        SqlCommand cmdKategori=new SqlCommand("Select distinct Kategori From Kitaplar Order By Kategori",conFriends);
        SqlDataReader drKategori; /* SqlDataReader nesnemizi tanımlıyoruz. Lütfen tanımlama şeklimize dikkat edin. Sanki bir
```

değişken tanımlıyoruz gibi. Herhangibir new yapıcı metodu yok. */

drKategori=cmdKategori.ExecuteReader(CommandBehavior.CloseConnection); /* SqlCommand nesnemizin ExecuteReader yardımıyla sorgumuzu çalıştırıyor ve sonuç kümesini temsil edecek SqlDataReader nesnemizi atıyoruz. Bu aşamada SqlDataReader nesnemiz sorgu sonucu oluşan veri kümesinin ilk satırının öncesine konumlanıyor. */

/* Bahsetmiş olduğumuz while döngüsü ile satırları teker teker ileri doğru olacak şekilde belleğe alıyoruz. Her bir t zamanında bir satır belleğe geliyor ve oradanda ListBox kontrolüne ilgili satırın, 0 indexli alanına ait değer GetString metodu ile alınıyor. */

```
while(drKategori.Read())
{
    IstKategori.Items.Add(drKategori.GetString(0));
}
```

drKategori.Close(); /* SqlDataReader nesnemiz kapatılıyor. Biz ExecuteReader metodunu çalıştırırken parametre olarak, CommandBehavior.CloseConnection değerini verdik. Bu bir SqlDataReader nesnesi kapatıldığında, açık olan bağlantının otomatik olarak kapatılmasını, yani SqlConnection bağlantısının otomatik olarak kapatılmasını sağlar. Buda system kaynaklarının serbest bırakılmasını sağlar.*/

```
}
}
private void btnKitaplar_Click(object sender, System.EventArgs e)
{
```

```
    string kategori=IstKategori.SelectedItem.ToString();
    baglantiAc();
    SqlCommand cmdKitaplar=new SqlCommand("Select distinct Adi,ID From Kitaplar Where Kategori='"+kategori+"' order by
Adi",conFriends);
```

```
    SqlDataReader drKitaplar;
    drKitaplar=cmdKitaplar.ExecuteReader(CommandBehavior.CloseConnection);
    int KitapSayisi=0;
    IstKitaplar.Items.Clear();
    while(drKitaplar.Read())
    {
        IstKitaplar.Items.Add(drKitaplar.GetString(0));
        KitapSayisi+=1;
    }
```

```
    drKitaplar.Close();
    lblKitapSayisi.Text=KitapSayisi.ToString();
```

/* Yukarıdaki döngüde elde edilen kayıt sayısını öğrenmek için KitapSayisi isimli bir sayacı döngü içine koyduğumuzu farketmişsinizdir. SqlDataReader nesnesi herhangi bir t zamanında bellekte sadece bir satırı temsil eder. Asıl veri kümesinin tamamını içermez, yani belleğe almaz. Bu nedenle veri kümesindeki satır sayısını temsil edecek, Count gibi bir metodu yoktur. İşte bu nedenle kayıt sayısını bu teknik ile öğrenmek durumundayız. */

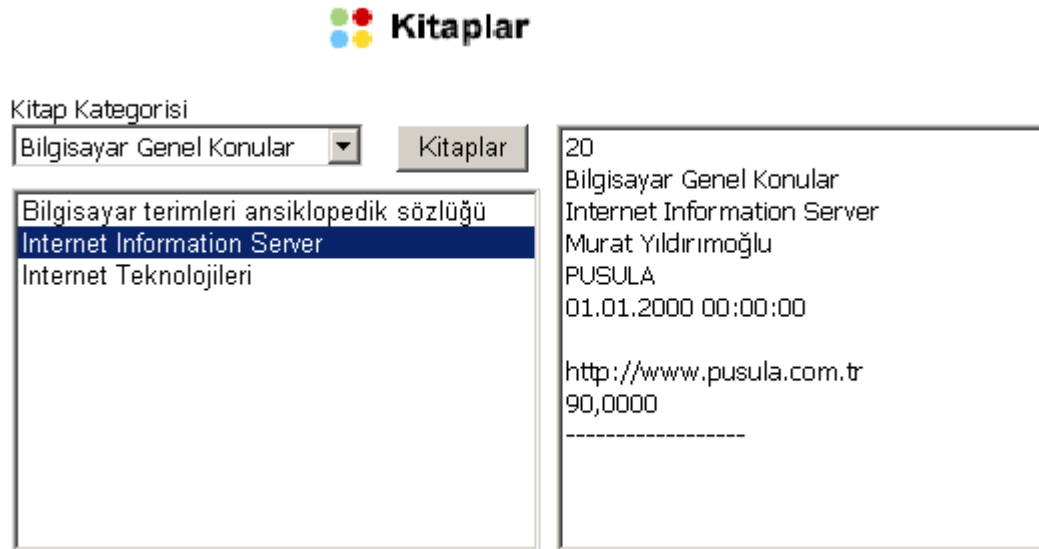
```
}
```

```

private void IstKitaplar_SelectedIndexChanged(object sender, System.EventArgs e)
{
    string adi=IstKitaplar.SelectedItem.ToString();
    baglantiAc();
    SqlCommand cmdKitapBilgisi=new SqlCommand("Select * From Kitaplar Where Adi='"+adi+"'","conFriends");
    SqlDataReader drKitapBilgisi;
    drKitapBilgisi=cmdKitapBilgisi.ExecuteReader(CommandBehavior.CloseConnection);
    IstKitapBilgisi.Items.Clear();
    while(drKitapBilgisi.Read())
    {
        /* FieldCount özelliği SqlDataReader nesnesinin t zamanında bellekte temsil etmiş olduğu satırın kolon sayısını vermektedir.
        Biz burada tüm kolonlardaki verileri okuyacak dolayısıyla t zamanında bellekte yer alan satırın verilerini elde edebileceğimiz bir For
        döngüsü oluşturduk. Herhangibir alanın değerine, drKitapBilgisi[i].ToString() ifadesi ile ulaşıyoruz. */
        for(int i=0;i<drKitapBilgisi.FieldCount;++i)
        {
            IstKitapBilgisi.Items.Add(drKitapBilgisi[i].ToString());
        }
        IstKitapBilgisi.Items.Add("-----");
    }
    drKitapBilgisi.Close();
}

```

Şimdi programımızı bir çalıştıralım ve sonuçlarını bir görelim.



3

Şekil 3. Programın çalışmasının sonucu.

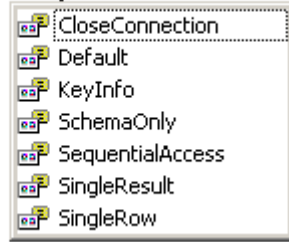
Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde SqlDataReader sınıfını incelemeye devam edeceğiz. Özellikle SqlCommand sınıfının ExecuteReader metodunun aldığı parameter değerlerine göre nasıl sonuçlar elde edebileceğimizi incelemeye çalışacağız. Bunun yanında SqlDataReader sınıfının diğer özelliklerindeki inceleyeceğiz. Hepinize mutlu günler dilerim.

SqlDataReader Sınıfı 2

Değerli Okurlarım, Merhabalar.

Bir önceki makalemizde SqlDataReader sınıfını incelemeye başlamıştık. Listeleme amaçlı veri kümelerinin görüntülenmesinde performans açısından etkin bir rol oynadığından bahsetmiştik. Bugünkü makalemizde, SqlDataReader sınıfının faydalı diğer özelliklerinden bahsedeceğiz. Öncelikle, bir SqlDataReader nesnesinin, geçerli ve açık bir SqlConnection nesnesi üzerinde çalışan bir SqlCommand nesnesi yardımıyla oluşturulduğunu hatırlayalım. Burada SqlCommand sınıfına ait ExecuteReader metodu kullanılmaktadır. ExecuteReader metoduna değişik parametreler geçirerek uygulamanın performansını dahada arttırabiliriz. Önceki makalemizde, CommandBehavior.CloseConnection parametre değerini kullanmıştık. CommandBehavior, çalıştırılacak olan sql sorgusu için bir davranış belirlememizi sağlar. SqlCommand nesnesinin ExecuteReader metodunun alabileceği parametre değerleri şekil 1 de görülmektedir. Bunların ne işe yaradığı kısaca tablo 1 'de bahsedilmiştir.

(CommandBehavior.)



Şekil 1. CommandBehavior Davranışları

CommandBehavior Değeri	İşlevi
CommandBehavior.CloseConnection	SqlDataReader nesnesi Close metodu ile kapatıldığında, ilişkili SqlConnection nesneside otomatik olarak kapatılır. Nitekim, işimiz bittiğinde SqlConnection nesnesinin açık unutulması sistem kaynaklarının gereksiz yere harcanmasına neden olur.
CommandBehavior.SingleRow	En çok kullanılan parametrelerden birisidir. Eğer sql sorgumuz tek bir satır döndürecek tipte ise bu davranış kullanmak performansı olumlu yönde etkiler. Örneğin PrimaryKey üzerinden yapılan sorgular. ("Select * From Tablo Where ID=3" tarzında.)

CommandBehavior.SingleResult	Tek bir değer döndürecek tipteki sorgular için kullanılır. Örneğin belli bir alandaki sayısal değerlerin toplamı veya tablodaki kayıt sayısını veren sorgular gibi. Bu tekniğe alternatif olan ve daha çok tercih edilen bir diğer yöntem, SqlCommand nesnesinin ExecuteScalar metodudur
CommandBehavior.SchemaOnly	Çalıştırılan sorgu sonucu elde edilen satır(satırların) sadece alan bilgisini döndürür.
CommandBehavior.SequentialAccess	Bazı durumlarda tablo alanları çok büyük boyutlu binary tipte veriler içerebilirler. Bu tarz büyük verilerinin okunması için en kolay yol bunları birer akım (stream) halinde belleğe okumak ve oradan ilgili nesnelere taşımaktır. SequentialAccess davranışı bu tarz akımların işlenmesine imkan tanırken performansıda arttırmaktadır.
CommandBehavior.KeyInfo	Bu durumda sql sorgusu sonucunda SqlDataReader nesnesi, tabloya ait anahtar alan bilgisini içerir.

Tablo 1. CommandBehavior Davranışları

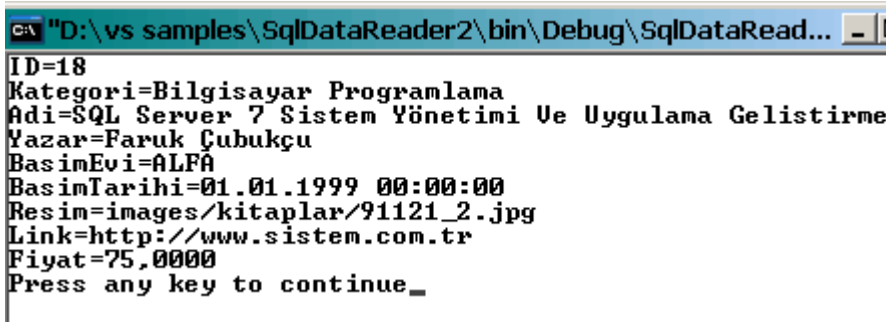
Şimdi dilerseniz basit Console uygulamaları ile, yukarıdaki davranışların işleyişlerini inceleyelim. CommandBehavior.CloseConnection durumunu önceki makalemizde işlediğimiz için tekrar işleme gereği duymuyorum. Şimdi en çok kullanacağımız davranışlardan birisi olan SingleRow davranışına bakalım. Uygulamamız ID isimli PrimaryKey alanı üzerinden bir sorgu çalıştırıyor. Dönen veri kümesinin tek bir satırdan oluşacağı kesindir. Bu durum, SingleRow davranışını kullanmak için en ideal durumdur.

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace SqlDataReader2
{
    class Class1
    {
        static void Main(string[] args)
        {
            SqlConnection conFriends=new SqlConnection("data source=localhost;integrated security=sspi;initial catalog=Friends");
            SqlCommand cmd=new SqlCommand("Select * From Kitaplar Where ID=18",conFriends); /* Sql sorgumuz ID isimli
primary key üzerinden bir sorgu çalıştırıyor ve 18 nolu ID değerine sahip satırı elde ediyor. Burada tek satırlık veri olduğu kesin. */
```

```

SqlDataReader dr;
conFriends.Open();
dr=cmd.ExecuteReader(CommandBehavior.SingleRow); /* Tek satırlık veri için davranışımızı SingleRow olarak
belirliyoruz. */
dr.Read(); /* Elde edilen satırı belleğe okuyoruz. Görüldüğü gibi herhangi bir while döngüsü kullanma gereği duymadık.*/
for(int i=0;i<dr.FieldCount;++i) /* Satırın alan sayısı kadar devam edecek bir döngü kuruyoruz ve her alanın adını
GetName, bu alanlara ait değerleride dr[i].ToString ile ekrana yazdırıyoruz. */
{
    Console.WriteLine(dr.GetName(i).ToString()+"="+dr[i].ToString());
}
dr.Close(); /* SqlDataReader nesnemizi kapatıyoruz. Ardından SqlConnection nesnemizide kapatmayı unutmuyoruz.
Böylece bu nesnelere ait kaynaklar serbest kalmış oluyor.*/
conFriends.Close();
}
}
}

```



Şekil 2. SingleRow davranışı.

Şimdi SingleResult davranışını inceleyelim. Bu kez Northwind veritabanında yer alan Products tablosundaki UnitPrice alanlarının ortalamasını hesaplayan bir sql sorgumuz var. Burada tek bir değer dönmektedir. İşte SingleResult bu duruma en uygun davranış olacaktır.

```

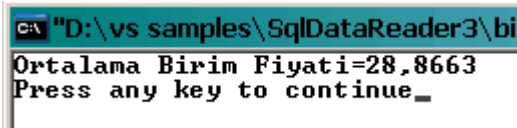
using System;
using System.Data;
using System.Data.SqlClient;
namespace SqlDataReader3
{
    class Class1
    {
        static void Main(string[] args)
        {

```

```

        SqlConnection conNorthwind=new SqlConnection("data source=localhost;integrated security=sspi;initial
catalog=Northwind");
        SqlCommand cmd=new SqlCommand("Select SUM(UnitPrice)/Count(UnitPrice)As [Ortalama Birim Fiyatı] From
Products",conNorthwind);
        SqlDataReader dr;
        conNorthwind.Open();
        dr=cmd.ExecuteReader(CommandBehavior.SingleResult);
        dr.Read();
        Console.WriteLine(dr.GetName(0).ToString()+"="+dr[0].ToString());
        dr.Close();
        conNorthwind.Close();
    }
}
}

```



Şekil 3. SingleResult davranışı.

Şimdi SchemaOnly davranışını inceleyelim. Önce aşağıdaki kodları yazıp çalıştıralım.

```

using System;
using System.Data;
using System.Data.SqlClient;
namespace SqlDataReader4
{
    class Class1
    {
        static void Main(string[] args)
        {
            SqlConnection conNorthwind=new SqlConnection("data source=localhost;integrated security=sspi;initial
catalog=Northwind");
            SqlCommand cmd=new SqlCommand("Select * From Products",conNorthwind);
            SqlDataReader dr;
            conNorthwind.Open();
            dr=cmd.ExecuteReader(CommandBehavior.SchemaOnly);
            dr.Read();
            try

```

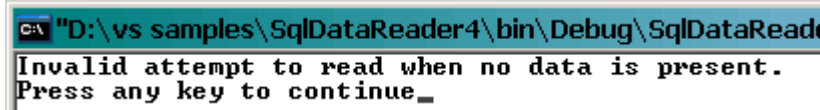


```

    {
        for(int i=0;i<dr.FieldCount;++i)
        {
            Console.WriteLine(dr.GetName(i).ToString()+" "+ dr.GetFieldType(i).ToString()+" "+dr[i].ToString());
        }
    }
    catch(Exception hata)
    {
        Console.WriteLine(hata.Message.ToString());
    }
    dr.Close();
    conNorthwind.Close();
}
}
}

```

Yukarıdaki console uygulamasını çalıştırdığımızda aşağıdaki hata mesajını alırız.

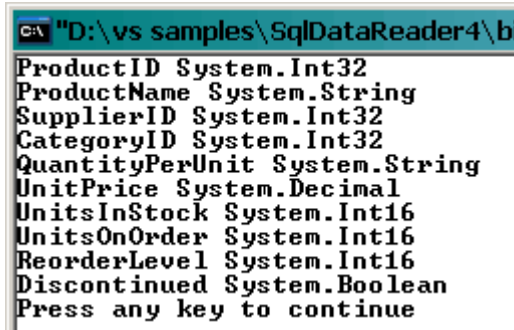


Şekil 4. Hata.

SchemaOnly davranışı sorgu ne olursa olsun sadece alan bilgilerini döndürür. Herhangibir veri döndürmez. Bu yüzden `dr[i].ToString()` ifadesi `i` nolu `indexe` sahip alan için herhangi bir veri bulamayacaktır. Kodun bu bölümünü aşağıdaki gibi değiştirirsek;

```
Console.WriteLine(dr.GetName(i).ToString()+" "+dr.GetFieldType(i).ToString());
```

Ve şimdi console uygulamamızı çalıştırsak aşağıdaki ekran görüntüsünü elde ederiz. `GetFieldType` metodu `i` indeksli alanın veri tipinin .NET'teki karşılığını döndürürken `GetName` ile bu alanın adını elde ederiz.



Şekil 5. SchemaOnly davranışı.

Şimdi SequentialAccess davranışını inceleyelim. Bu sefer pubs isimli veritabanında yer alan pub_info isimli tablonun Text tipinde uzun veriye sahip pr_info alanının kullanacağız. Sorgumuz belli bir pub_id değerine sahip satırın pr_info alanını alıyor. Biz GetChars metodunu kullanarak alan içindeki veriyi karakter karakter okuyor ve beleğe doğru bir akım (stream) oluşturuyoruz. GetChars metodu görüldüğü üzere 5 parametre almaktadır. İl parametre ile hangi alanın okunacağını belirtiriz. İkin paramtere bu alanın kaçınıcı karakterinden itibaren okunmaya başlanacağı bildirir. Üçüncü parameter ise char tipinden bir diziyi belirtir. Okunan her bir karakter bu diziye aktarılacaktır. Dördüncü parameter ile dizinin kaçınıcı elemanından itibaren aktarımın yapılacağı belirtilir. Son parametremiz ise ne kadar karakter okunacağını belirtmektedir. Şimdi kodlarımızı yazalım.

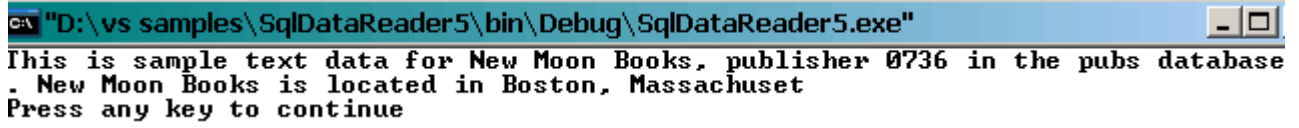
```
using System;
using System.Data;
using System.Data.SqlClient;
namespace SqlDataReader5
{
    class Class1
    {
        static void Main(string[] args)
        {
            SqlConnection conPubs=new SqlConnection("data source=localhost;integrated security=sspi;initial catalog=pubs");
            SqlCommand cmd=new SqlCommand("Select pr_info From pub_info where pub_id=0736",conPubs);
            SqlDataReader dr;
            conPubs.Open();
            dr=cmd.ExecuteReader(CommandBehavior.SequentialAccess);
            dr.Read();
            try
            {
                char[] dizi=new char[130]; /* 130 char tipi elemandan oluşan bir dizi tanımladık. */
                dr.GetChars(0,0,dizi,0,130); /* Dizimize pr_info alanından 130 karakter okuduk.*/
                for(int i=0;i<dizi.Length;++i) /* Dizideki elemanları ekrana yazdırıyoruz. */
                {
                    Console.Write(dizi[i]);
                }
                Console.WriteLine();
            }
            catch(Exception hata)
            {
                Console.WriteLine(hata.Message.ToString());
            }
            dr.Close();
            conPubs.Close();
        }
    }
}
```

```

    }
}
}

```

Sonuç olarak ekran görüntümüz aşağıdaki gibi olacaktır.



Şekil 6. SequentialAccess davranışı.

Evet değerli Okurlarım. Geldik bir makalemizin daha sonuna. Umarım sizi, SqlDataReader sınıfı ile ilgili bilgilerle donatabilmişimdir. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Boxing (Kutulamak) ve Unboxing (Kutuyu Kaldırmak)

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, Boxing ve Unboxing kavramlarını incelemeye çalışacağız. Boxing değer türü bir değişkeni, referans türü bir nesneye aktarmaktır. Unboxing işlemi ise bunun tam tersidir. Yani referans türü değişkenin işaret ettiği değeri tekrar , değer türü bir değişkene aktarmaktır. Bu tanımlarda karşımıza çıkan ve bilmemiz gereken en önemli noktalar, değer türü değişkenler ile referans türü nesnelerin bellekte tutuluş şekilleridir.

Net ortamında iki tür veri tipi vardır. Referans tipleri (reference type) ve değer tipleri (value type). İki veri türünün bellekte farklı şekillerde tutulmaları nedeni ile boxing ve unboxing işlemleri gündeme gelmiştir. Bu nedenle öncelikle bu iki farklı veri tipinin bellekte tutuluş şekillerini iyice anlamamız gerekmektedir.

Bu anlamda karşımıza iki önemli bellek bölgesi çıkar. Yığın (stack) ve öbek (heap). Değer tipi değişkenler, örneğin bir integer değişken vb.. belleğin stack adı verilen kısmında tutulurlar. DotNet'te yer alan değer türleri aşağıdaki tabloda yer almaktadır. Bu tiplerin stack bölgesinde nasıl tutulduğuna ilişkin açıklayıcı şekli de aşağıda görebilirsiniz.

Value type (Değer Tipleri)	
bool	long
byte	sbyte
char	short
decimal	Struct (Yapılar)

double	uint
Enum (Numaralandırıcılar)	ulong
float	ushort
int	

Tablo 1. Değer Tipleri

Memory (Bellek)



Şekil 1. Değer Tiplerinin Bellekte Tutuluşu

Şekildende görüldüğü gibi, Değer Türleri bellekte, Stack dediğimiz bölgede tutulurlar. Şimdi buraya kadar anlaşılmayan bir şey yok. İlginç olan reference(başvuru) tiplerinin bellekte nasıl tutulduğudur. Adındanda anlaşıldığı gibi reference tipleri asıl veriye bir başvuru içerirler. Örneğin sınıflardan türettiğimiz nesneler bu tiplerdendir. Diğer başvuru tipleri ise aşağıdaki tabloda yer almaktadır.

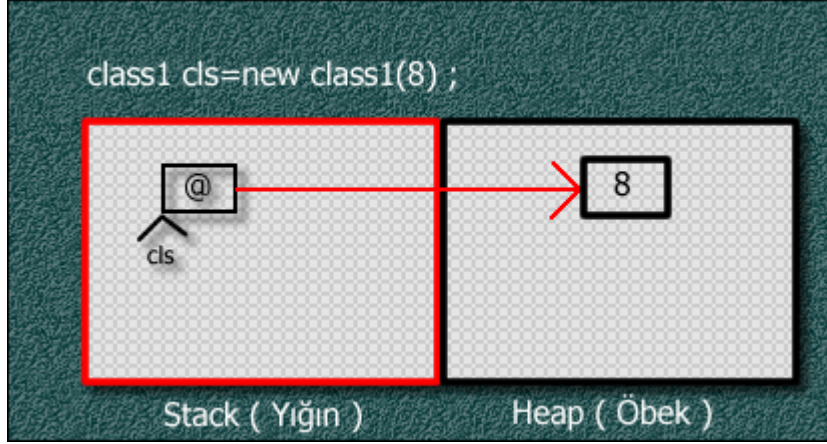
Reference Type (Başvuru Tipleri)
Class (sınıflar)
Interface (arayüzler)
Delegate (delegeler)

Object
String

Tablo 2. Başvuru Tipleri

Şimdi gelin başvuru türlerinin bellekte nasıl tutulduklarına bakalım.

Memory (Bellek)



Şekil 2. Başvuru tiplerinin bellekte tutuluşu.

Görüldüğü gibi ,başvuru tipleri hakikatende isimlerinin layığını vermekteler. Nitekim asıl veriler öbek'te tutulurken yığında bu verilere bir başvuru yer almaktadır. İki veri türü arasındaki bu farktan sonra bir farkta bu verilerle işlemiz bittiğinde geri iade ediliş şekilleridir. Değer türleri ile işlemiz bittiğinde bunların yığında kapladıkları alanlar otomatik olarak yığına geri verilir. Ancak referans türlerinde sadece yığındaki başvuru sisteme geri verilir. Verilerin tutulduğu öbekteki alanlar, Garbage Collector'un denetimindedirler ve ne zaman sisteme iade edilecekleri tam olarak bilinmez. Bu ayrı bir konu olmakla beraber oldukça karmaşıktır. İlerleyen makalelerimizde bu konudan da bahsetmeye çalışacağım.

Değer türleri ile başvuru türleri arasındaki bu temel farktan sonra gelelim asıl konumuza. . NET'te her sınıf aslında en üst sınıf olan Object sınıfından türer. Yani her sınıf aslında System.Object sınıfından kalıtım yolu ile otomatik olarak türetilmiş olur. Sorun object gibi referans bir türe, değer tipi bir değerın aktarılmasında yaşanır. . NET'te herşey aslında birer nesne olarak düşünülebilir. Bir değer türünü bir nesneye atamaya çalıştığımızda, değer türünün içerdiği verinin bir kopyasının yığından alınıp, öbeğe taşınması ve nesnenin bu veri kopyasına başvurması gerekmektedir. İşte bu olay kutulama (boxing) olarak adlandırılır. Bu durumu minik bir örnek ile inceleyelim.

```
using System;
namespace boxunbox
{
    class Class1
    {
```

```

static void Main(string[] args)
{
    double db=509809232323;
    object obj;
    obj=db;
    Console.WriteLine(db.ToString());
    Console.WriteLine(obj.ToString());
    db+=1;
    Console.WriteLine(db.ToString());
    Console.WriteLine(obj.ToString());
}
}
}

```

Kodumuzun çalışmasını inceleyelim. Db isimli double değişkenimiz bir değer tipidir. Örnekte bu değer tipini object tipinden bir nesneye aktarıyoruz. Bu halde bu değerler içerisindeki verileri ekrana yazdırıyoruz. Sonra db değerimizi 1 arttırıyor ve tekrar bu değerlerin içeriğini ekrana yazdırıyoruz. İşte sonuç;

```

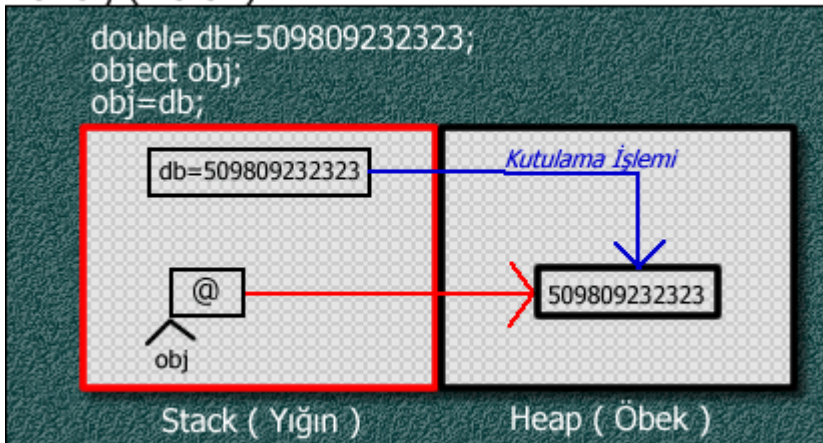
C:\ "D:\vs samples\boxunbox\bir
509809232323
509809232323
509809232324
509809232323
Press any key to continue_

```

Şekil 3 Boxing İşlemi

Görüldüğü gibi db değişkenine yapılan arttırım object türünden obj nesnemize yansımamıştır. Çünkü boxing işlemi sonucu, obj nesnesi, db değerinin öbekteki kopyasına başvurmaktadır. Oysaki artım db değişkeninin yığında yer alan orjinal değeri üzerinde gerçekleşmektedir. Bu işlemi açıklayan şekil aşağıda yer almaktadır.

Memory (Bellek)

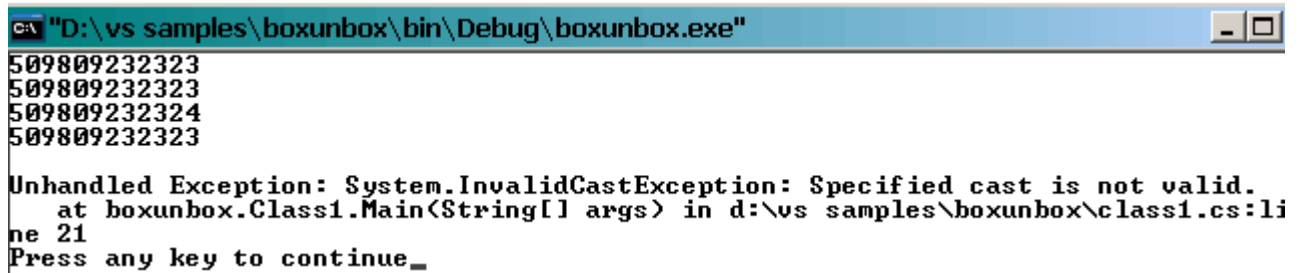


Şekil 4. Boxing İşlemi

Boxing işlemi otomatik olarak yapılan bir işlemdir. Ancak UnBoxing işleminde durum biraz daha değişir. Bu kez , başvuru nesneminin işaret ettiği veriyi öbekten alıp yığındaki bir değer tipi alanı olarak kopyalanması söz konusudur. İşte burada tip uyumsuzluğu denen bir kavramla karşılaşırız. Öbekten , yığına kopyalanacak olan verinin, yığında kendisi için ayrılan yerin aynı tipte olması veya öbekteki tipi içerebilecek tipte olması gerekmektedir. Örneğin yukarıdaki örneğimize unboxing işlemini uygulayalım. Bu kez integer tipte bir değer türüne atama gerçekleştirelim.

```
using System;
namespace boxunbox
{
    class Class1
    {
        static void Main(string[] args)
        {
            double db=509809232323;
            object obj;
            obj=db;
            Console.WriteLine(db.ToString());
            Console.WriteLine(obj.ToString());
            db+=1;
            Console.WriteLine(db.ToString());
            Console.WriteLine(obj.ToString());
            int intDb;
            intDb=(int)obj;
            Console.WriteLine(intDb.ToString());
        }
    }
}
```

Bu kodu çalıştırdığımızda InvalidCastException istisnasının fırlatılacağını göreceksiniz. Çünkü referenans tipimizin öbekte başvurduğu veri tipi integer bir değer için fazla büyüktür. Bu noktada (int) ile açıkça dönüşümü bildirmiş olsak dahi bu hatayı alırız.



```
C:\ "D:\vs samples\boxunbox\bin\Debug\boxunbox.exe"
509809232323
509809232323
509809232324
509809232323
Unhandled Exception: System.InvalidCastException: Specified cast is not valid.
   at boxunbox.Class1.Main<String[] args> in d:\vs samples\boxunbox\class1.cs:line 21
Press any key to continue_
```

Şekil 5. InvalidCastException İstisnası

Ancak küçük tipi, büyük tipe dönüştürmek gibi bir serbestliğimiz vardır. Örneğin,

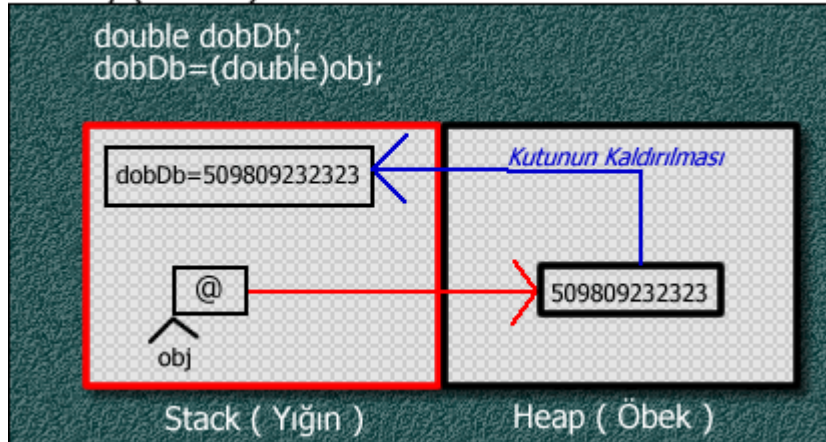
```

using System;
namespace boxunbox
{
    class Class1
    {
        static void Main(string[] args)
        {
            double db=509809232323;
            object obj;
            obj=db;
            Console.WriteLine(db.ToString());
            Console.WriteLine(obj.ToString());
            db+=1;
            Console.WriteLine(db.ToString());
            Console.WriteLine(obj.ToString());
            /*int intDb;
            intDb=(int)obj;
            Console.WriteLine(intDb.ToString());*/
            double dobDb;
            dobDb=(double)obj;
            Console.WriteLine(dobDb.ToString());
        }
    }
}

```

Bu durumda kodumuz sorunsuz çalışacaktır. Çünkü yığında yer alan veri tipi daha büyük boyutlu bir değer türünün içine koyulabilir. İşte buradaki aktarım işlemi unboxing olarak isimlendirilmiştir. Yani boxing işlemi ile kutulanmış bir veri kümesi, öbekten alınıp tekrar yığındaki bir Alana konulmuş dolayısıyla kutudan çıkartılmıştır. Olayın grafiksel açıklaması aşağıdaki gibidir.

Memory (Bellek)



Şekil 6. Unboxing İşlemi

Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Çok Kanallı(Multithread) Uygulamalar

Değerli Okurlarım, Merhabalar.

Bugünkü makelemiz ile birlikte threading kavramını en basit haliyle tanımaya çalışacağız. Sonraki makalelerimizde de threading kavramını daha üst seviyede işlemeye çalışacağız.

Bugün hepimiz bilgisayar başındayken aynı anda pek çok uygulamanın sorunsuz bir şekilde çalıştığını görürüz. Bir belge yazarken, aynı zamanda müzik dinleyebilir, internet üzerinden program indirebilir ve sistemimizin kaynaklarının elverdiği ölçüde uygulamayla eşzamanlı olarak çalışabiliriz. Bu bize, günümüz işlemcilerinin ve üzerlerinde çalışan işletim sistemlerinin ne kadar yetenekli olduğunu gösterir. Gösterir mi acaba?

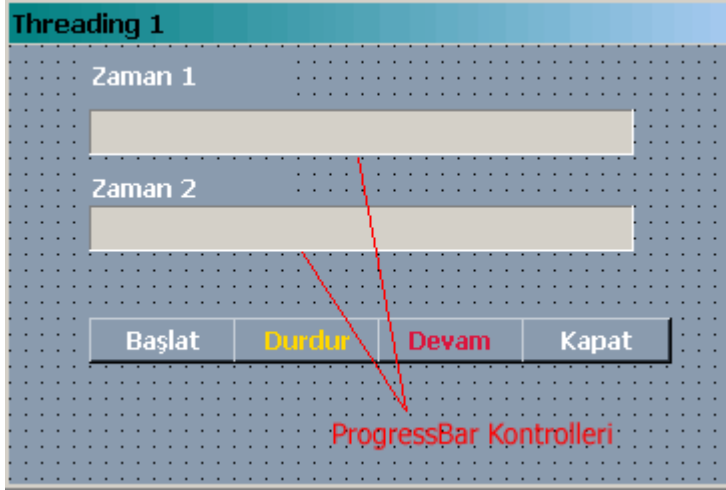
Aslında tek işlemcili makineler günümüzün modern sihirbazları gibidirler. Gerçekte çalışan uygulamaların tüm işlemleri aynı anda gerçekleşmemektedir. Fakat işlemciler öylesine büyük saat hızlarına sahiptirlerki. İşlemcinin yaptığı, çalıştırılan uygulamaya ait işlemleri iş parçacıkları(thread) halinde ele almaktır. Her bir iş parçacığı bir işlemin birden fazla parçaya bölünmesinden oluşur. İşlemciler her iş parçacığı için bir zaman dilimi belirler. T zaman diliminde bir işlem parçacığı yürütülür ve bu zaman dilim bittiğinde işlem parçacığı geçici bir süre için durur. Ardından kuyrukta bekleyen diğer iş parçacığı başka bir zaman dilimi içinde çalıştırılır. Bu böylece devam ederken, işlemcimiz her iş parçacığına geri döner ve tüm iş parçacıkları sıra sıra çalıştırılır. Dedik ya, işlemciler bu işlemleri çok yüksek saat ve frekans hızında gerçekleştirir. İşte bu yüksek hız nedeniyle tüm bu olaylar saniyenin milyon sürelerinde gerçekleşir ve sanki tüm bu uygulamalar aynı anda çalışıyor hissi verir.

Gerçektende uygulamaları birbirleriyle paralel olarak ve eş zamanlı çalıştırmak aslında birden fazla işlemciye sahip sistemler için gereklenir.

Bugünkü uygulamamız ile, bahsetmiş olduğumuz threadin kavramına basit bir giriş yapacağız. Nitekim threading kavramı ve teknikleri, uygulamalarda profesyonel olarak kod yazmayı gerektirir. Daha açık şekilde söylemek gerekirse bir uygulama içinde yazdığımız kodlara uygulayacağımızı thread'ler her zaman avantaj sağlamaz. Bazı durumlarda dezavantaja dönüşüp programların daha yavaş çalışmasına neden olabilir. Nitekim thread'lerin çalışma mantığını iyi kavramak ve uygulamalarda titiz davranmak gerekir.

Örneğin thread'lerin zaman dilimlerine bölündüklerinde sistemin nasıl bir önceki veya daha önceki thread'i çalıştırabildiğini düşünelim. İşlemci zaman dilimini dolduran bir thread için donanımda bir kesme işareti bırakır, bunun ardında thread'e ait bir takım bilgiler belleğe yazılır ve sonra bu bellek bölgesinde Context adı verilen bir veri yapısına depolanır. Sistem bu thread'e döneceği zaman Context'te yer alan bilgilere bakar ve hangi donanımın kesme sinyali verdiğini bulur. Ardından bu sinyal açılır ve işlemin bir sonraki işlem parçacığının çalışacağı zaman dilimine girilir. Eğer thread işlemini çok fazla kullanırsanız bu durumda bellek kaynakları da fazlası ile tüketmiş olursunuz. Bu thread'leri neden titiz bir şekilde programlamamız gerektiğini anlatan nedenlerden sadece birisidir. Öyleki yanlış yapılan thread programlamaları sistemlerin kilitlenmesine de yol açacaktır.

Threading gördüğünüz gibi çok basit olmayan bir kavramdır. Bu nedenle olayı daha iyi açıklayabileceğimi düşündüğüm örneklerime geçmek istiyorum. Uygulamamızın formu aşağıdaki şekildeki gibi olucak.



Şekil 1. Form Tasarımımız.

Şimdi kodlarımızı yazalım.

```
public void z1()
{
    for(int i=1;i<60;++i)
    {
        zaman1.Value+=1;
        for(int j=1;j<10000000;++j)
        {
            j+=1;
        }
    }
}

public void z2()
{
    for(int k=1;k<100;++k)
    {
        zaman2.Value+=1;
        for(int j=1;j<25000000;++j)
        {
            j+=1;
        }
    }
}

private void btnBaslat_Click(object sender, System.EventArgs e)
{

```

```

z1();
z2();
}

```

Program kodlarımızı kısaca açıklayalım. Z1 ve z2 isimli metodlarımız progressBar kontrollerimizin değerlerini belirli zaman aralıklarıyla arttırıyorlar. Bu işlemleri gerçekleştirmek için, Başlat başlıklı butonumuza tıklıyoruz. Burada önce z1 daha sonrada z2 isimli metodumuz çalıştırılıyor. Bunun sonucu olarak önce zaman1 isimli progressBar kontrolümüz doluyor ve dolması bittikten sonra zaman2 isimli progressBar kontrolümüzün value değeri arttırılarak dolduruluyor.

Şimdi bu programın şöyle çalışmasını istediğimizi düşünelim. Her iki progressBar'da aynı anda dolmaya başlasınlar. İsteddiğimiz zaman z1 ve z2 metodlarının çalışmasını durduralım ve tekrar başlatabilelim. Tekrar başlattığımızda ise progressBar'lar kaldıkları yerden dolmaya devam etsinler. Söz ettiğimiz aslında her iki metodunda aynı anda çalışmasıdır. İşte bu işi başarmak için bu metodları sisteme birer iş parçacığı (thread) olarak tanıtmalı ve bu thread'leri yönetmeliyiz.

.Net ortamında thread'ler için System.Threading isim uzayını kullanırız. Öncelikle programımıza bu isim uzayını ekliyoruz. Ardından z1 ve z2 metodlarını birer iş parçacığı olarak tanımlamamız gerekiyor. İşte kodlarımız.

```

public void z1()
{
    for(int i=1;i<60;++i)
    {
        zaman1.Value+=1;
        for(int j=1;j<10000000;++j)
        {
            j+=1;
        }
    }
}

public void z2()
{
    for(int k=1;k<100;++k)
    {
        zaman2.Value+=1;
        for(int j=1;j<25000000;++j)
        {
            j+=1;
        }
    }
}

ThreadStart ts1;
ThreadStart ts2;

```

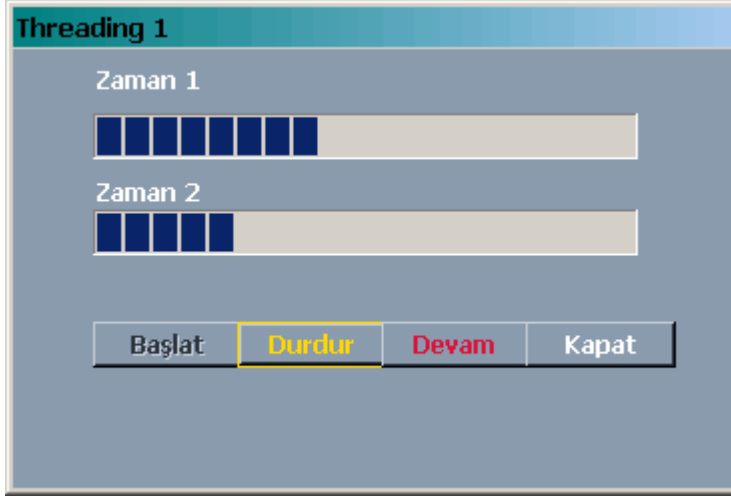
```

Thread t1;
Thread t2;
private void btnBaslat_Click(object sender, System.EventArgs e)
{
    ts1=new ThreadStart(z1); /* ThreadStart iş parçacığı olarak kullanılacak metod için bir temsilcidir. Bu metod için tanımlanacak
thread sınıfı nesnesi için paramtere olucak ve bu nesnenin hangi metodu iş parçacığı olarak göreceğini belirtecektir. */
    ts2=new ThreadStart(z2);
    t1=new Thread(ts1);
    t2=new Thread(ts2);
    t1.Start(); /* İş parçacığını Start metodu ile başlatıyoruz. */
    t2.Start();
    btnBaslat.Enabled=false;
}
private void btnDurdur_Click(object sender, System.EventArgs e)
{
    t1.Suspend(); /* İş parçacığı geçici bir süre için uyku moduna geçer. Uyku modundaki bir iş parçacığını tekrar aktif hale
getirmek için Resume metodu kullanılır. */
    t2.Suspend();
}
private void btnDevam_Click(object sender, System.EventArgs e)
{
    t1.Resume(); /* Uyku modundaki iş parçacığının kaldığı yerden devam etmesini sağlar. */
    t2.Resume();
}
private void btnKapat_Click(object sender, System.EventArgs e)
{
    if(t1.IsAlive) /* Eğer iş parçacıkları henüz sonlanmamışsa bunlar canlıdır ve IsAlive özellikleri true değerine sahiptir.
Programımızda ilk biten iş parçacığı t1 olduğundan onun bitip bitmediğini kontrol ediyoruz. Eğer bitmiş ise programımız close
metodu sayesinde kapatılabilir. */
    {
        MessageBox.Show("Çalışan threadler var program sonlanamaz.");
    }
    else
    {
        this.Close();
    }
}
}

```

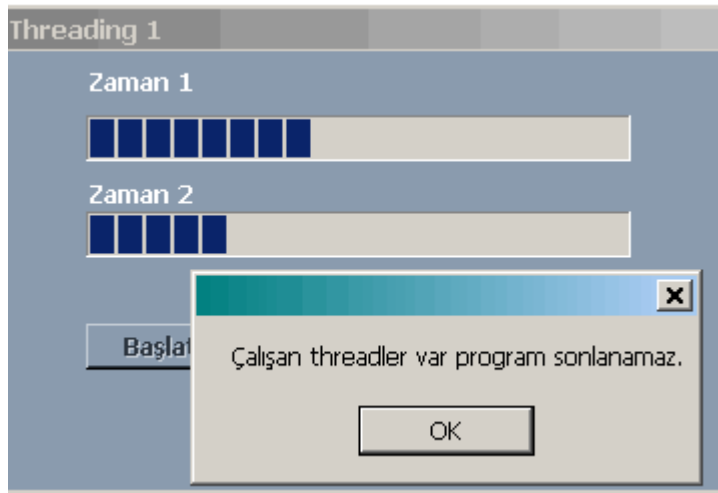
Uygulamamızda z1 ve z2 isimli metodlarımızı birer iş parçacığı (thread) haline getirdik. Bunun için System.Threading isim uzayında yer alan ThreadStart ve Thread sınıflarını kullandık. ThreadStart sınıfı , iş parçacığı olucak metodu temsil eden bir delegate gibi

davranır. İş parçacıklarını başlatacak(start), durduracak(suspend), devam ettirecek(resume) thread nesnelerimizi tanımladığımız yapıcı metod ThreadStart sınıfından bir temsilciyi parametre olarak alır. Sonuç itibarıyla kullanıcı Başlat başlıklı button kontrolüne tıkladığında, her iki progressBar kontrolünde aynı zamanda dolmaya başladığını ve ilerlediklerini görürüz. Bu aşamada Durdur isimli button kontrolüne tıklarsak her iki progressBar'ın ilerleyişinin durduğunu görürüz. Nitekim iş parçacıklarının Suspend metodu çağırılmış ve metodların çalıştırılması durdurulmuştur.



Şekil 2. Suspend metodu sonrası.

Bu andan sonra tekrar Devam button kontrolüne tıklarsak thread nesnelerimiz Resume metodu sayesinde çalışmalarına kaldıkları yerden devam edeceklerdir. Dolayısıyla progressBar kontrollerimizde kaldıkları yerden dolmaya devam ederler. Bu sırada programı kapatmaya çalışmamız henüz sonlanmamış iş parçacıkları nedeni ile hataya neden olur. Bu nedenle Kapat button kontrolünde IsAlive özelliği ile iş parçacıklarının canlı olup olmadığı yani metodların çalışmaya devam edip etmediği kontrol edilir. Eğer sonlanmamışsa kullanıcı aşağıdaki mesaj kutusu ile uyarılır.



Şekil 3. İş Parçacıkları henüz sonlanmamış ise.

Evet geldik Threading ile ilgili makale dizimizin ilk bölümünün sonuna . Bir sonraki makalemizde Threading kavramını dahada derinlemesine incelemeye çalışacağız. Hepinize mutlu günler dilerim.

StreamReader Sınıfı Yardımıyla Dosya Okumak

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, sistemimizde yer alan text tabanlı dosyaları nasıl okuyabileceğimizi incelemeye çalışacağız. .NET ortamında, dosyaların okunması için streamler(akımlar) kullanılır. Bugün işleyeceğimi StreamReader sınıfı bunlardan bir tanesidir.

StreamReader sınıfı dosyaların okunmasını, dosyalara yazılmasını vb.. sağlar. StreamReader sınıfını bir FileStream nesnesi ile kullanabileceğimiz gibi, tek başınada kullanabiliriz. Kullanabileceğimiz yapıcı metodlardan birisi;

```
public StreamReader( Stream stream );
```

dir. Bu yapıcı metod Stream tipinden bir nesne alır. Bu stream nesnesi çoğunlukla FileStream sınıfından türetilmiş bir nesne olur. Bu yapıcı metodumuz dışında direkt olarak okuma amacı ile StreamReader nesnesini ;

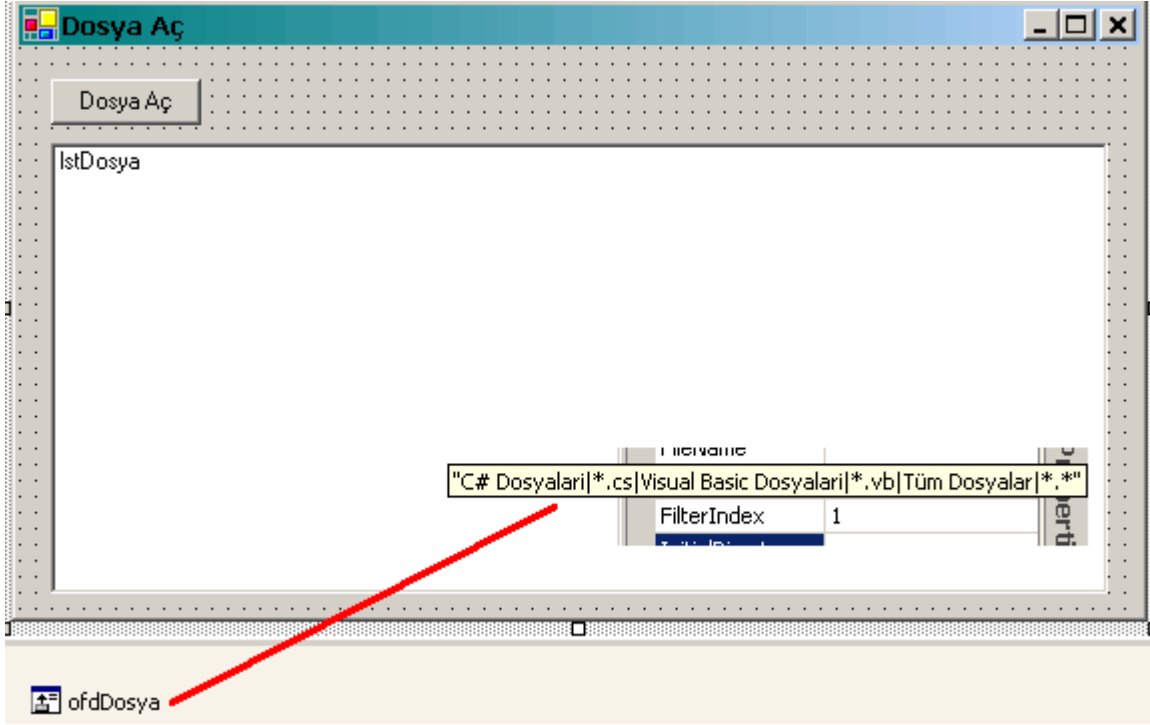
```
public StreamReader( string path );
```

yapıcısı ilede oluşturabiliriz. Burada string tipindeki path değişkenimiz, okumak amacıyla açacağımız dosyanın tam adresini temsil etmektedir. StreamReader nesnesi ile dosyamızı açtıktan sonra dosya içindeki verileri ReadLine metodu ile okuyabiliriz. ReadLine metodu, dosyadan her defasında bir satır okur ve bunun string olarak geriye döndürür. Metodun prototipi aşağıdaki gibidir.

```
public override string ReadLine();
```

Genellikle bu metod bir while döngüsü ile kullanılır. Bu sayede dosyanın tüm içeriğinin okunması sağlanmış olur. ReadLine metodu geriye null değerini döndürdüğü zaman dosyanın sonuna gelindiği anlaşılır. Bu nedenle While döngüsünde kontrol ifadesi okunan her bir satırın null olup olmadığına bakar.

Şimdi bu kısa açıklamaların ardından derseniz uygulamamızı yazalım. Bu küçük uygulamamızda kullanıcının seçmiş olduğu bir dosyayı bir listBox kontrolüne satır bazında açacağız.Öncelikle aşağıdaki örnek formu oluşturalım. OpenFileDialog kontrolümüzün Filter özelliğinede ekranda görülen değerleri aktarıyoruz. Böylece OpenFileDialog kontrolümüz açıldığında cs,vb uzantılı dosyaları ve tüm dosyaları görebileceğiz.



Şekil 1. Form Tasarımımız.

Şimdi dilersenin projemizin kodlarını yazalım.

```
private void btnDosyaAc_Click(object sender, System.EventArgs e)
```

```
{
```

```
    if(ofdDosya.ShowDialog()==DialogResult.OK)
```

```
    {
```

```
        this.Text=ofdDosya.FileName.ToString();
```

```
        FileStream d=new FileStream(ofdDosya.FileName,FileMode.Open,FileAccess.Read); /* Burada okuma amacı ile
        OpenFileDialog kontrolünden seçtiğimiz dosyaya bir akım oluşturuyoruz. */
```

```
        StreamReader sr=new StreamReader(d); /* Şimdi ise StreamReader nesnemizi FileStream nesnesini kullanarak
        oluşturuyoruz. */
```

```
        String input;
```

```
        /* Bu döngü, sr isimli StreamReader nesnemizin temsil ettiği akım vasıtasıyla, dosyamızdan bir satır alır ve bunu bellekteki
        tampon bölgeye yerleştirir. Daha sonra bunu bir string değişkene atıyoruz. Ardından bunun değerinin null olup olmadığına
        bakıyoruz. Null olması halinde dosya sonuna geldiğimiz anlaşılmaktadır. ReadLine metodu otomatik olarak bir satırı tampona
        aldıktan sonra, akımdan bir sonraki satırı okur ve belleğe alır. */
```

```
        while ((input=sr.ReadLine())!=null)
```

```
        {
```

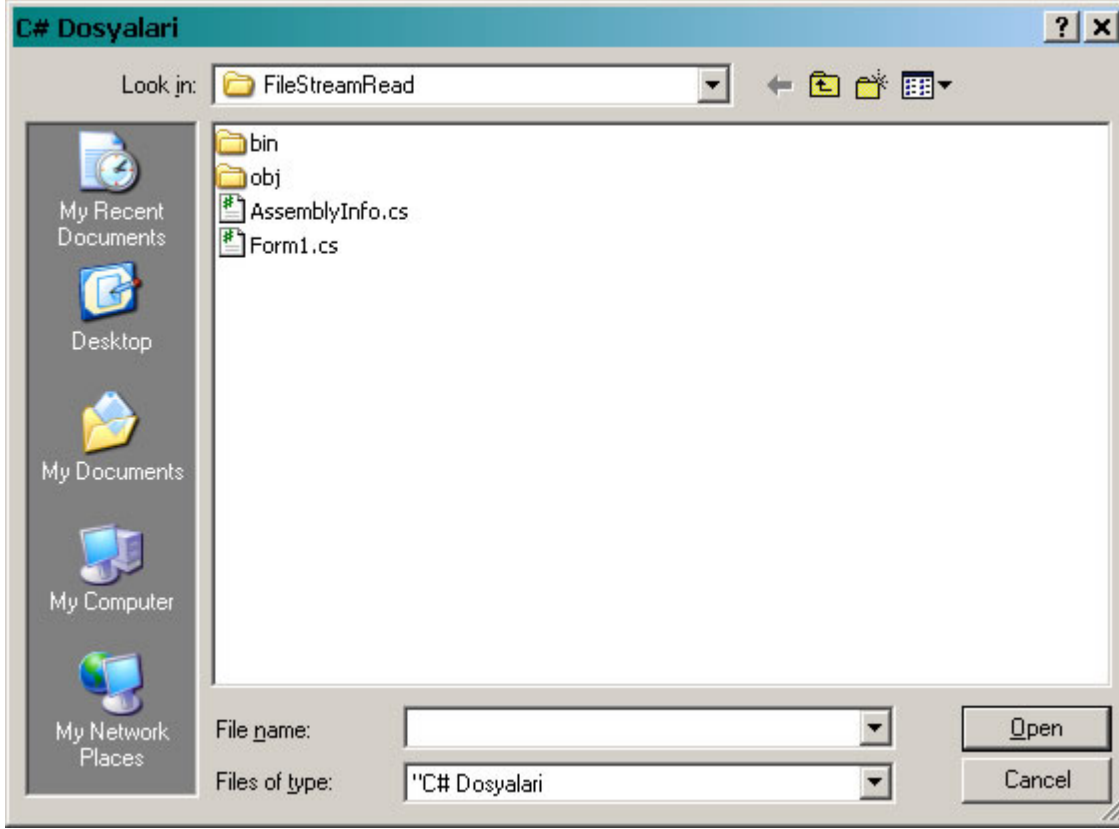
```
            IstDosya.Items.Add(input); /* ListBox kontrolümüze oonan her bir satırı ekliyoruz.*/
```

```
        }
```

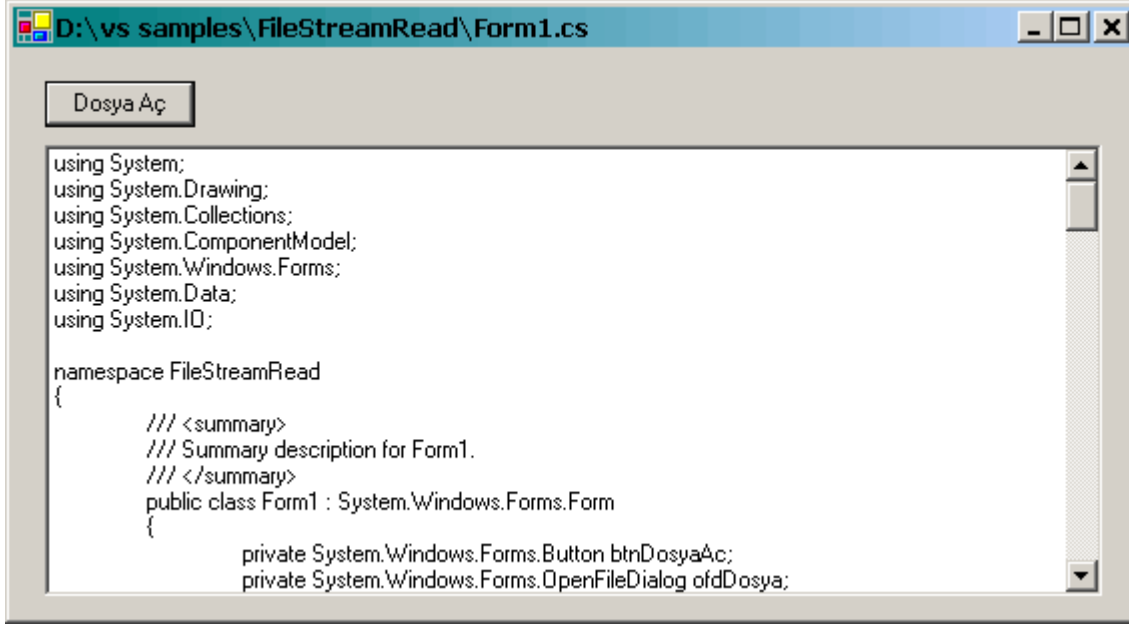
```
        sr.Close(); /* Son olarak StreamReader nesnemizi ve FileStream nesnemizi kapatıyoruz. */
```

```
d.Close();  
}  
}
```

Şimdi uygulamamızı çalıştıralım. Görüldüğü gibi OpenFileDialog kutumuzda Filter özelliğinde belirlediğimiz dosyalar görünüyor.



Şekil 2. OpenFileDialog İletişim Kutumuz.



Şekil 3. Dosyamızın içeriği.

Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Thread'leri Belli Süreler Boyunca Uyutmak ve Yoketmek

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde iş parçacıklarının belli süreler boyunca nasıl durgunlaştırılabileceğini yani etkisizleştirebileceğimizi işlemey çalışcaz. Ayrıca iş parçacıklarının henüz sonlanmadan önce nasıl yokedildiklerini göreceğiz.

Bir önceki makalemizde hatırlayacak olursanız, iş parçacıkları haline getirdiğimiz metodlarımızda işlemleri yavaşlatmak amacı ile bazı döngüler kullanmıştık. Gerçek hayatta çoğu zaman, iş parçacıklarının belirli süreler boyunca beklemesini ve süre sona erdiğinde tekrardan işlemlerine kaldığı yerden devam etmesini istediğimiz durumlar olabilir. Önceki makalemizde kullandığımız Suspend metodu ile ilgili iş parçacığını durdurabiliyorduk. Bu ilgili iş parçacıklarını geçici süre ile bekletmenin yollarından birisidir. Ancak böyle bir durumda bekletilen iş parçacığını tekrar hareketlendirmek kullanıcının Resume metodunu çalıştırması ile olabilir. Oysaki biz, iş parçacığımızın belli bir süre boyunca beklemsini isteyebiliriz. İşte böyle bir durumda Sleep metodunu kullanırız. Bu metodun iki adet overload edilmiş versiyonu vardır.

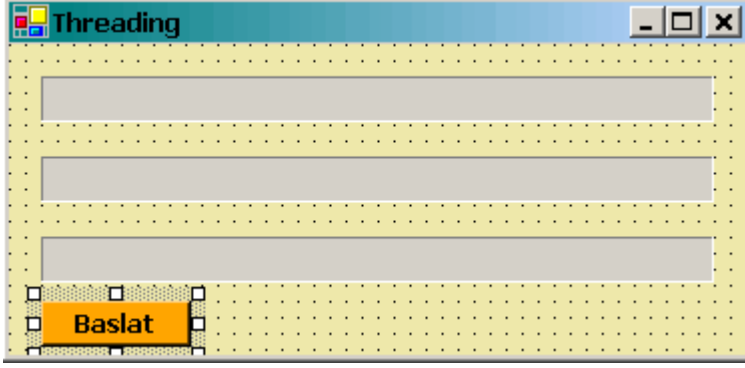
```
public static void Sleep(int);
```

```
public static void Sleep(TimeSpan);
```

Biz bugünkü uygulamamızda ilk versiyonu kullanacağız. Bu versiyonda metodumuz parametre olarak int tipinde bir değer almaktadır. Bu değer milisaniye cinsinden süreyi bildirir. Metodun Static bir metod olduğu dikkatinizi çekmiş olmalıdır. Static bir metod olması nedeni ile, Sınıf adı ile birlikte çağırılmak zorundadır. Yani herhangi bir thread nesnesinin ardından Sellp metodunu yazamassınız. Peki o halde bekleme süresinin hangi iş parçacığı için geçerli olacağını nereden bileceğiz. Bu nedenle, bu metod iş parçacığı olarak tanımlanan metod blokları içerisinde kullanılır. Konuyu örnek üzerinden inceleyince daha iyi anlayacağız. Metod çalıştırıldığında parametresinde belirtilen süre boyunca geçerli iş parçacığını bekletir. Bu bekleme diğer parçacıkların çalışmasını

engellemez. Süre sona erince, iş parçacığımız çalışmasına devam edecektir. Şimdi derseniz örnek bir uygulama geliştirelim ve konuya açıklık getirmeye çalışalım.

Formumuzda bu kez üç adet ProgressBar kontrolümüz var. Baslat başlıklı düğmeye bastığımızda iş parçacıklarımız çalışıyor ve tüm ProgressBar'lar aynı anda değişik süreler ile ilerliyor. Burada iş parçacıkları olarak belirlediğimiz metodlarda kullandığımız Sleep metodlarına dikkat edelim. Tabi kodlarımızı yazmadan önce System.Threading isim uzayını eklemeyi unutmayalım.



Şekil 1. Form Tasarımımız.

```
public void pb1Ileri()
{
    for(int i=1;i<100;++i)
    {
        pb1.Value+=1;
        Thread.Sleep(800);
    }
}

public void pb2Ileri()
{
    for(int i=1;i<100;++i)
    {
        pb2.Value+=1;
        Thread.Sleep(500); /* Metodumuz iş parçacığı olarak başladıktan sonra döngü içince her bir artımdan sonra 500
milisaniye bekler. */
    }
}

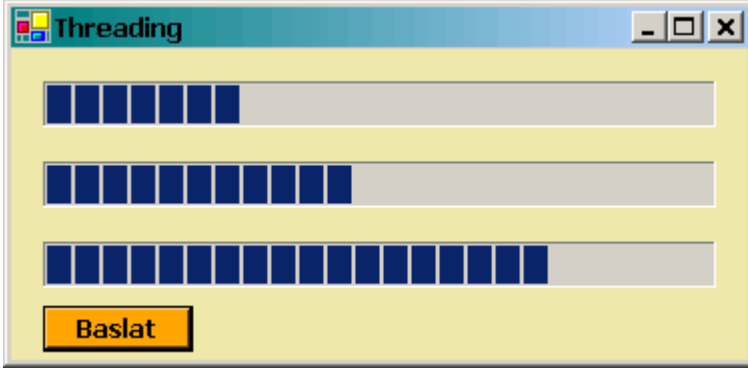
public void pb3Ileri()
{
    for(int i=1;i<100;++i)
    {
        pb3.Value+=1;
        Thread.Sleep(300);
    }
}
```

```

    }
}
/* ThreadStart temsilcilerimiz ve Thread nesnelerimizi tanımlıyoruz. */
ThreadStart ts1;
ThreadStart ts2;
ThreadStart ts3;
Thread t1;
Thread t2;
Thread t3;
private void btnBaslat_Click(object sender, System.EventArgs e)
{
    /* ThreadStart temsilcilerimizi ve Thread nesnelerimizi oluşturuyoruz. */
    ts1=new ThreadStart(pb1Ileri);
    ts2=new ThreadStart(pb2Ileri);
    ts3=new ThreadStart(pb3Ileri);
    t1=new Thread(ts1);
    t2=new Thread(ts2);
    t3=new Thread(ts3);
    /* Thread nesnelerimizi start metodu ile başlatıyoruz. */
    t1.Start();
    t2.Start();
    t3.Start();
}

```

Uygulamamızı çalıştıralım. Her iş parçacığı Sleep metodu ile belirtilen süre kadar beklemeler ile çalışmasına devam eder. Örneğin pb3Ileri metodunda iş parçacığımız ProgressBar'ın Value değerini her bir arttırdıktan sonra 300 milisaniye bekler ve döngü bir sonraki değerden itibaren devam eder. Sleep metodu ile Suspend metodları arasında önemli bir bağ daha vardır. Bildiğiniz gibi Suspend metodu ilede bir iş parçacığını durdurabilmekteyiz. Ancak bu iş parçacığını tekrar devam ettirmek için Resume metodunu kullanmamız gerekiyor. Bu iki yöntem arasındaki fark idi. Diğer önemli olgu ise şudur; bir iş parçacığı metodu içinde, Sleep metodunu kullanmış olsak bile, programın herhangi bir yerinden bu iş parçacığı ile ilgili Thread nesnesinin Suspend metodunu çağırdığımızda, bu iş parçacığı yine duracaktır. Bu andan itibaren Sleep metodu geçerliliğini, bu iş parçacığı için tekrardan Resume metodu çağırılincaya kadar kaybedecektir. Resume çağırısından sonra ise Sleep metodları yine işlemeye devam eder.



Şekil 2. Sleep Metodunun Çalışması

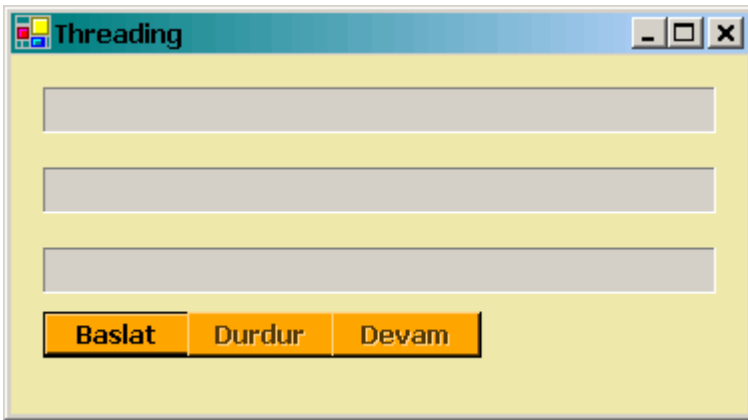
Şimdi gelelim diğer konumuz olan bir iş parçacığının nasıl yok edileceğine. Bir iş parçacığını yoketmek amacı ile Abort metodunu kullanabiliriz. Bu metod çalıştırıldığında derleyici aslında bir ThreadAbortException istisnası üretir ve iş parçacığını yoketmeye zorlar. Abort yöntemi çağırıldığında, ilgili iş parçacığını tekrar resume gibi bir komutla başlatamayız. Diğer yandan Abort metodu iş parçacığı ile ilgili metod için ThreadAbortException istisnasını fırlattığında (throw) , bu metod içinde bir try..catch..finally korumalı bloğunda bu istisnayı yakalayabiliriz veya Catch bloğunda hiç bir şey yazmasa ise program kodumuz kesilmeden çalışmasına devam edecektir.

Abort metodu ile bir iş parçacığı sonlandırıldığında, bu iş parçacığını Start metodu ile tekrar çalıştırmak istersek;

"ThreadStateException' Additional information: Thread is running or terminated; it can not restart."

hatasını alırız. Yani iş parçacığımızı tekrar baştan başlatmak gibi bir şansımız yoktur.

Şimdi bu metodu inceleyeceğimiz bir kod yazalım. Yukarıdaki uygulamamızı aşağıdaki şekilde geliştirelim.



Şekil 3. Form Tasarımımız.

Kodlarımıza geçelim.

```
public void pb1Ileri()
{
```

```
try
{
    for(int i=1;i<100;++i)
    {
        pb1.Value+=1;
        Thread.Sleep(800);
    }
}
catch(ThreadAbortException hata)
{
}
finally
{
}
}

public void pb2Ileri()
{
    for(int i=1;i<100;++i)
    {
        pb2.Value+=1;
        Thread.Sleep(500); /* Metodumuz iş parçacığı olarak başladıktan sonra döngü içince her bir artımdan sonra 500
milisaniye bekler. */
    }
}

public void pb3Ileri()
{
    for(int i=1;i<100;++i)
    {
        pb3.Value+=1;
        Thread.Sleep(300);
    }
}

/* ThreadStart temsilcilerimiz ve Thread nesnelerimizi tanımlıyoruz. */
ThreadStart ts1;
ThreadStart ts2;
ThreadStart ts3;
Thread t1;
Thread t2;
Thread t3;
```

```

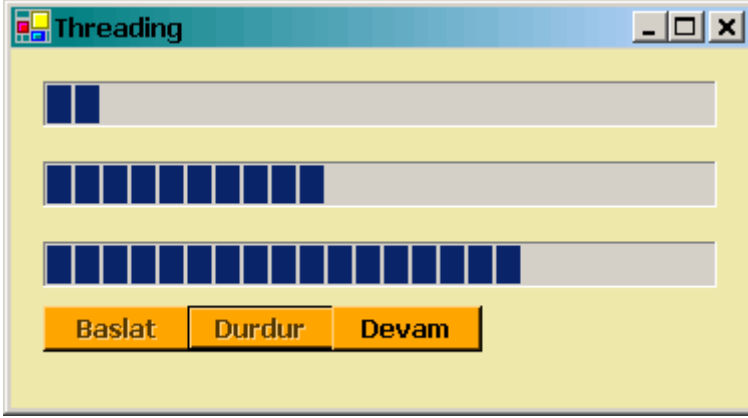
private void btnBaslat_Click(object sender, System.EventArgs e)
{
    /* ThreadStart temsilcilerimizi ve Thread nesnelerimizi oluşturunuz. */
    ts1=new ThreadStart(pb1Ileri);
    ts2=new ThreadStart(pb2Ileri);
    ts3=new ThreadStart(pb3Ileri);
    t1=new Thread(ts1);
    t2=new Thread(ts2);
    t3=new Thread(ts3);
    /* Thread nesnelerimizi start metodu ile başlatıyoruz. */
    t1.Start();
    t2.Start();
    t3.Start();
    btnBaslat.Enabled=false;
    btnDurdur.Enabled=true;
    btnDevam.Enabled=false;
}

private void btnDurdur_Click(object sender, System.EventArgs e)
{
    t1.Abort(); /* t1 isimli Thread'imizi yok ediyoruz. Dolayısıyla pb1Ileri isimli metodumuzunda çalışmasını sonlandırmış oluyoruz.
    */
    /* Diğer iki iş parçacığını uyutuyoruz. */
    t2.Suspend();
    t3.Suspend();
    btnDurdur.Enabled=false;
    btnDevam.Enabled=true;
}

private void btnDevam_Click(object sender, System.EventArgs e)
{
    /* İş parçacıklarını tekrar kaldıkları yerden çalıştırıyoruz. İşte burada t1 thread nesnesini Resume metodu ile tekrar kaldığı
    yerden çalıştıramayız. Bu durumda programımız hataya düşecektir. Nitekim Abort metodu ile Thread'imiz sonlandırılmıştır. Aynı
    zamanda Start metodu ile Thread'imizi başta başlatamayız.*/
    t2.Resume();
    t3.Resume();
    btnDurdur.Enabled=true;
    btnDevam.Enabled=false;
}

```

Uygulamamızı deneyelim.



Şekil 4. Uygulamanın çalışması sonucu.

Değerli okurlarım geldik bir makalemizin daha sonuna. Bir sonraki makalemizde Threading konusunu işlemeye devam edeceğiz. Hepinize mutlu günler dilerim.

Thread'lerde Öncelik(Priority) Durumları

Değerli Okurlarım, Merhabalar.

İş parçacıklarını işlediğimiz yazı dizimizin bu üçüncü makalesinde, iş parçacıklarının birbirlerine karşı öncelik durumlarını incelemeye çalışacağız. İş parçacıkları olarak tanımladığımız metodların çalışma sıralarını, sahip oldukları öneme göre değiştirmek durumunda kalabiliriz. Normal şartlar altında, oluşturduğumuz her bir iş parçacığı nesnesi aynı ve eşit önceliğe sahiptir. Bu öncelik değeri Normal olarak tanımlanmıştır. Bir iş parçacığının önceliğini değiştirmek istediğimizde, Priority özelliğinin değerini değiştiririz. Priority özelliğinin .NET Framework'teki tanımı aşağıdaki gibidir.

public ThreadPriority Priority {get; set;}

Özellikimiz ThreadPriority numaralandırıcısı (enumerator) tipinden değerler almaktadır. Bu değerler aşağıdaki tabloda verilmiştir.

Öncelik Değeri
Highest
AboveNormal
Normal
BelowNormal
Lowest

Tablo 1. Öncelik(Priority) Değerleri

Programlarımızı yazarken, iş parçacıklarının çalışma şekli verilen öncelik değerlerine göre değişecektir. Elbette tahmin edeceğimiz gibi yüksek öncelik değerlerine sahip olan iş parçacıklarının işaret ettikleri metodlar diğerlerine göre daha sık aralıklarda çağırılacak, dolayısıyla düşük öncelikli iş parçacıklarının referans ettiği metodlar daha geç sonlanacaktır. Şimdi olayı daha iyi canlandırabilmek için aşağıdaki örneğimizi geliştirelim. Daha önceden söylediğimiz gibi, bir iş parçacığının Priority özelliğine her hangibir değer vermez isek, standart olarak Normal kabul edilir. Buda tüm iş parçacıklarının varsayılan olarak eşit önceliklere sahip olacağını anlamına gelmektedir. Şimdi aşağıdaki formumuzu oluşturalım. Uygulamamız iki iş parçacığına sahip. Bu parçacıkların işaret ettiği metodlardan birisi 1' den 1000' e kadar sayıp bu değerleri bir label kontrolüne yazıyor. Diğeri ise 1000' den 1' e kadar sayıp bu değerleri başka bir label kontrolüne yazıyor. Formumuzun görüntüsü aşağıdakine benzer olmalıdır.

Şekil 1. Form Tasarımımız.

Şimdide program kodlarımızı yazalım.

/* Bu metod 1' den 1000' e kadar sayar ve değerleri lblSayac1 isimli label kontrolüne yazar.*/

```
public void Say1()
```

```
{
```

```
    for(int i=1;i<1000;++i)
```

```
    {
```

```
        lblSayac1.Text=i.ToString();
```

```
        lblSayac1.Refresh(); /* Refresh metodu ile label kontrolünün görüntüsünü tazeleriz. Böylece herbir i değerinin label kontrolünde görülebilmesini sağlamış oluyoruz. */
```

```
        for(int j=1;j<900000000;++j)
```

```
        {
```

```
            j+=1;
```

```
        }
```

```
    }
```



```

}
/* Bu metod 1000' den 1' e kadar sayar ve değerleri lblSayac2 isimli label kontrolüne yazar.*/
public void Say2()
{
    for(int i=1000;i>=1;i--)
    {
        lblSayac2.Text=i.ToString();
        lblSayac2.Refresh();
        for(int j=1;j<45000000;++j)
        {
            j+=1;
        }
    }
}

/* ThreadStart ve Thread nesnelerimizi tanımlıyoruz. */
ThreadStart ts1;
ThreadStart ts2;
Thread t1;
Thread t2;
private void btnBaslat1_Click(object sender, System.EventArgs e)
{
    /* Metodlarımızı ThreadStart nesneleri ile ilişkilendiriyoruz ve ThreadStart nesnelerimizi oluşturuyoruz.*/
    ts1=new ThreadStart(Say1);
    ts2=new ThreadStart(Say2);
    /* İş parçacıklarımızı, ilgili metodların temsil eden ThreadStart nesnelerimiz ile oluşturuyoruz.*/
    t1=new Thread(ts1);
    t2=new Thread(ts2);
    /* İş parçacıklarımızı çalıştırıyoruz.*/
    t1.Start();
    t2.Start();
    btnBaslat1.Enabled=false;
    btnIptal.Enabled=true;
}

private void btnIptal_Click(object sender, System.EventArgs e)
{
    /* İş parçacıklarımızı iptal ediyoruz. */
    t1.Abort();
    t2.Abort();
    btnBaslat1.Enabled=true;
}

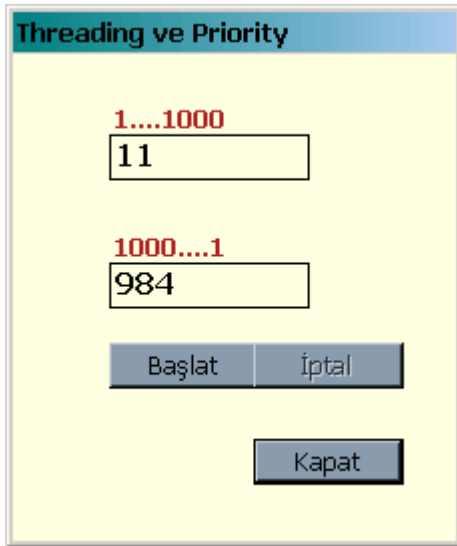
```

```

    btnIptal.Enabled=false;
}
private void btnKapat_Click(object sender, System.EventArgs e)
{
    /* Uygulamayı kapatmak istediğimizde, çalışan iş parçacığı olup olmadığını kontrol ediyoruz. Bunun için iş parçacıklarının IsAlive özelliğinin değerlerine bakıyoruz. Nitekim kullanıcının, herhangi bir iş parçacığı sonlanmadan uygulamayı kapatmasını istemiyoruz. Ya iptal etmeli yada sonlanmalarını beklemeli. İptal ettiğimizde yani Abort metodları çalıştırıldığında hatırlayacağınız gibi, iş parçacıklarının IsAlive değerleri false durumuna düşüyordu, yani iptal olmuş oluyorlardı.*/
    if(!t1.IsAlive) && (!t2.IsAlive)
    {
        Close();
    }
    else
    {
        MessageBox.Show("Hala kapatılamamış iş parçacıkları var. Lütfen bir süre sonra tekrar deneyin.");
    }
}
}

```

Uygulamamızda şu an için bir yenilik yok aslında. Nitekim iş parçacıklarımız için bir öncelik ayarlaması yapmadık. Çünkü size göstermek istediğim bir husus var. Bir iş parçacığı için herhangi bir öncelik ayarı yapmadığımızda bu değer varsayılan olarak Normal dir. Dolayısıyla her iş parçacığı eşit önceliğe sahiptir. Şimdi örneğimizi çalıştıralım ve kafamıza göre bir yerde iptal edelim.



Şekil 2. Öncelik değeri Normal.

Ben 11 ye 984 değerinde işlemi iptal ettim. Tekrar iş parçacıklarını Başlat başlıklı butona tıklayıp çalıştırsak ve yine aynı yerde işlemi iptal edersek, ya aynı sonucu alırız yada yakın değerleri elde ederiz. Nitekim programımızı çalıştırdığımızda arka planda çalışan işletim sistemine ait pek çok iş parçacığıda çalışma sonucunu etkiler. Ancak aşağı yukarı aynı veya yakın değerle ulaşırız.

Oysa bu iş parçacıklarının öncelik değerlerini değiştirdiğimizde sonuçların çok daha farklı olabileceğini söyleyebiliriz. Bunu daha iyi anlayabilmek için örneğimizi geliştirelim ve iş parçacıklarının öncelik değerleri ile oynayalım. Formumuzu aşağıdaki gibi tasarlayalım.

Şekil 3. Formumuzun yeni tasarımı.

Artık iş parçacıklarını başlatmadan önce önceliklerini belirleyeceğiz ve sonuçlarını incelemeye çalışacağız. Kodlarımızı şu şekilde değiştirelim. Önemli olan kod satırlarımız, iş parçacıklarının Priority özelliklerinin değiştiği satırlardır.

/* Bu metod 1' den 1000' e kadar sayar ve değerleri lblSayac1 isimli label kontrolüne yazar.*/

```
public void Say1()
```

```
{
```

```
    for(int i=1;i<1000;++i)
```

```
    {
```

```
        lblSayac1.Text=i.ToString();
```

```
        lblSayac1.Refresh(); /* Refresh metodu ile label kontrolünün görüntüsünü tazeleriz. Böylece herbir i değerinin label kontrolünde görülebilmesini sağlamış oluyoruz. */
```

```
        for(int j=1;j<90000000;++j)
```

```
        {
```

```
            j+=1;
```

```
        }
```

```
    }
```

```
}
```

/* Bu metod 1000' den 1' e kadar sayar ve değerleri lblSayac2 isimli label kontrolüne yazar.*/

```
public void Say2()
```

```
{
```

```
    for(int i=1000;i>=1;i--)
```

```
    {
```

```

        lblSayac2.Text=i.ToString();
        lblSayac2.Refresh();
        for(int j=1;j<45000000;+=j)
        {
            j+=1;
        }
    }
}

ThreadPriority tp1; /* Priority öncelikleri ThreadPriority tipindedirler. */
ThreadPriority tp2;
/* OncelikBelirle metodu, kullanıcının TrackBar'da seçtiği değerleri göz önüne alarak, iş parçacıklarının Priority özelliklerini belirlemektedir. */
public void OncelikBelirle()
{
    /* Switch ifadelerinde, TrackBar kontrollünün değerine göre , ThreadPriority değerleri belirleniyor. */
    switch(tbOncelik1.Value)
    {
        case 1:
        {
            tp1=ThreadPriority.Lowest; /* En düşük öncelik değeri. */
            break;
        }
        case 2:
        {
            tp1=ThreadPriority.BelowNormal; /* Normalin biraz altı. */
            break;
        }
        case 3:
        {
            tp1=ThreadPriority.Normal; /* Normal öncelik değeri. Varsayılan değer budur.*/
            break;
        }
        case 4:
        {
            tp1=ThreadPriority.AboveNormal; /* Normalin biraz üstü öncelik değeri. */
            break;
        }
        case 5:
        {

```

```
        tp1=ThreadPriority.Highest; /* En üst düzey öncelik değeri. */
        break;
    }
}
switch(tbOncelik2.Value)
{
    case 1:
    {
        tp2=ThreadPriority.Lowest;
        break;
    }
    case 2:
    {
        tp2=ThreadPriority.BelowNormal;
        break;
    }
    case 3:
    {
        tp2=ThreadPriority.Normal;
        break;
    }
    case 4:
    {
        tp2=ThreadPriority.AboveNormal;
        break;
    }
    case 5:
    {
        tp2=ThreadPriority.Highest;
        break;
    }
}
/* İş Parçacıklarımıza öncelik değerleri aktarılıyor.*/
t1.Priority=tp1;
t2.Priority=tp2;
}
/* ThreadStart ve Thread nesnelerimizi tanımlıyoruz. */
ThreadStart ts1;
ThreadStart ts2;
```

```

Thread t1;
Thread t2;
private void btnBaslat1_Click(object sender, System.EventArgs e)
{
    /* Metodlarımızı ThreadStart nesneleri ile ilişkilendiriyoruz ve ThreadStart nesnelerimizi oluşturuyoruz.*/
    ts1=new ThreadStart(Say1);
    ts2=new ThreadStart(Say2);
    /* İş parçacıklarımızı, ilgili metodların temsil eden ThreadStart nesnelerimiz ile oluşturuyoruz.*/
    t1=new Thread(ts1);
    t2=new Thread(ts2);
    OncelikBelirle(); /* Öncelik ( Priority ) değerleri, iş parçacıkları Start metodu ile başlatılmadan önce belirlenmelidir. */
    /* İş parçacıklarımızı çalıştırıyoruz.*/
    t1.Start();
    t2.Start();
    btnBaslat1.Enabled=false;
    btnIptal.Enabled=true;
    tbOncelik1.Enabled=false;
    tbOncelik2.Enabled=false;
}
private void btnIptal_Click(object sender, System.EventArgs e)
{
    /* İş parçacıklarımızı iptal ediyoruz. */
    t1.Abort();
    t2.Abort();
    btnBaslat1.Enabled=true;
    btnIptal.Enabled=false;
    tbOncelik1.Enabled=true;
    tbOncelik2.Enabled=true;
}
private void btnKapat_Click(object sender, System.EventArgs e)
{
    /* Uygulamayı kapatmak istediğimizde, çalışan iş parçacığı olup olmadığını kontrol ediyoruz. Bunun için iş parçacıklarının
    IsAlive özelliğinin değerlerine bakıyoruz. Nitekim kullanıcının, herhangi bir iş parçacığı sonlanmadan uygulamayı kapatmasını
    istemiyoruz. Ya iptal etmeli yada sonlanmalarını beklemeli. İptal ettiğimizde yani Abort metodları çalıştırıldığında hatırlayacağınız
    gibi, iş parçacıklarının IsAlive değerleri false durumuna düşüyordu, yani iptal olmuş oluyorlardı.*/
    if(!t1.IsAlive) && (!t2.IsAlive))
    {
        Close();
    }
}

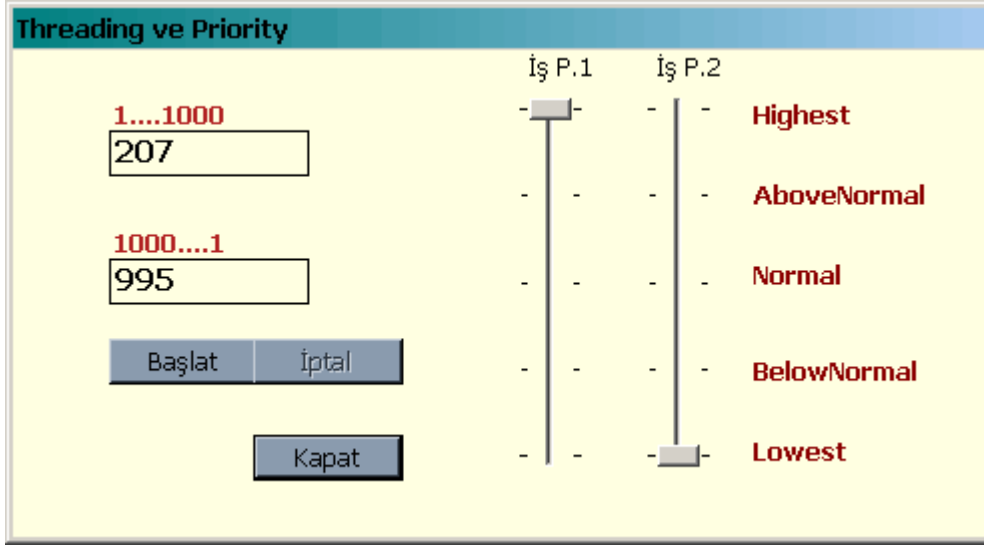
```

```

else
{
    MessageBox.Show("Hala kapatılamamış iş parçacıkları var. Lütfen bir süre sonra tekrar deneyin.");
}
}

```

Şimdi örneğimizi çalıştıralım ve birinci iş parçacığımız için en yüksek öncelik değerini (Highest) ikinci iş parçacığımız içinde en düşük öncelik değerini (Lowest) seçelim. Sonuçlar aşağıdakine benzer olacaktır.



Şekil 4. Önceliklerin etkisi.

Görüldüğü gibi öncelikler iş parçacıklarının çalışmasını oldukça etkilemektedir. Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde iş parçacıkları hakkında ilerlemeye devam edeceğiz. Görüşmek dileğiyle hepinize mutlu günler dilerim.

İşe Yarar Bir MultiThreading(Çok Kanallı) Uygulama Örneği

Değerli Okurlarım, Merhabalar.

Bundan önceki üç makalemizde iş parçacıkları hakkında bilgiler vermeye çalıştım. Bu makalemde ise işimize yarayacak tarzda bir uygulama geliştirecek ve bilgilerimizi pekiştireceğiz. Bir iş parçacığının belkide en çok işe yarayacağı yerlerden birisi veritabanı uygulamalarıdır. Bazen programımız çok uzun bir sonuç kümesi döndürecek sorgulara veya uzun sürecek güncelleme ifadeleri içeren sql cümlelerine sahip olabilir. Böyle bir durumda programın diğer öğeleri ile olan aktivitemizi devam ettirebilmek isteyebiliriz. Ya da aynı anda bir den fazla iş parçacığında, birden fazla veritabanı işlemini yaptırarak bu işlemlerin tamamının daha kısa sürelerde bitmesini sağlayabiliriz. İşte bu gibi nedenleri göz önüne alarak bu gün birlikte basit ama faydalı olacağına inandığım bir uygulama geliştireceğiz.

Olayı iyi anlayabilmek için öncelikle bir milat koymamız gerekli. İş parçacığından önceki durum ve sonraki durum şeklinde. Bu nedenle uygulamamızı önce iş parçacığı kullanmadan oluşturacağız. Sonrada iş parçacığı ile. Şimdi programımızdan kısaca bahsedelim. Uygulamamız aşağıdaki sql sorgusunu çalıştırıp, bellekteki bir DataSet nesnesinin referans ettiği bölgeyi, sorgu sonucu dönen veri kümesi ile dolduracak.

SELECT Products.* From [Order Details] Cross Join Products

Bu sorgu çalıştırıldığında, Sql sunucusunda yer alan Northwind veritabanı üzerinden, 165936 satırlık veri kümesi döndürür. Elbette normalde böyle bir işlemi istemci makinenin belleğine yığmamız anlamsız. Ancak sunucu üzerinde çalışan ve özellikle raporlama amacı ile kullanılan sorguların bu tip sonuçlar döndürmeside olasıdır. Şimdi bu sorguyu çalıştırıp sonuçları bir DataSet'e alan ve bu veri kümesini bir DataGrid kontrolü içinde gösteren bir uygulama geliştirelim. Öncelikle aşağıdaki formumuzu tasarlayalım.

Şekil 1. Form Tasarımımız.

Şimdide kodlarımızı yazalım.

```
DataSet ds;
public void Bagla()
{
    dataGrid1.DataSource=ds.Tables[0];
}
public void Doldur()
{
    SqlConnection conNorthwind=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=sspi");
    conNorthwind.Open();
    SqlDataAdapter daNorthwind=new SqlDataAdapter("SELECT Products.* From [Order Details] Cross Join
Products",conNorthwind);
```

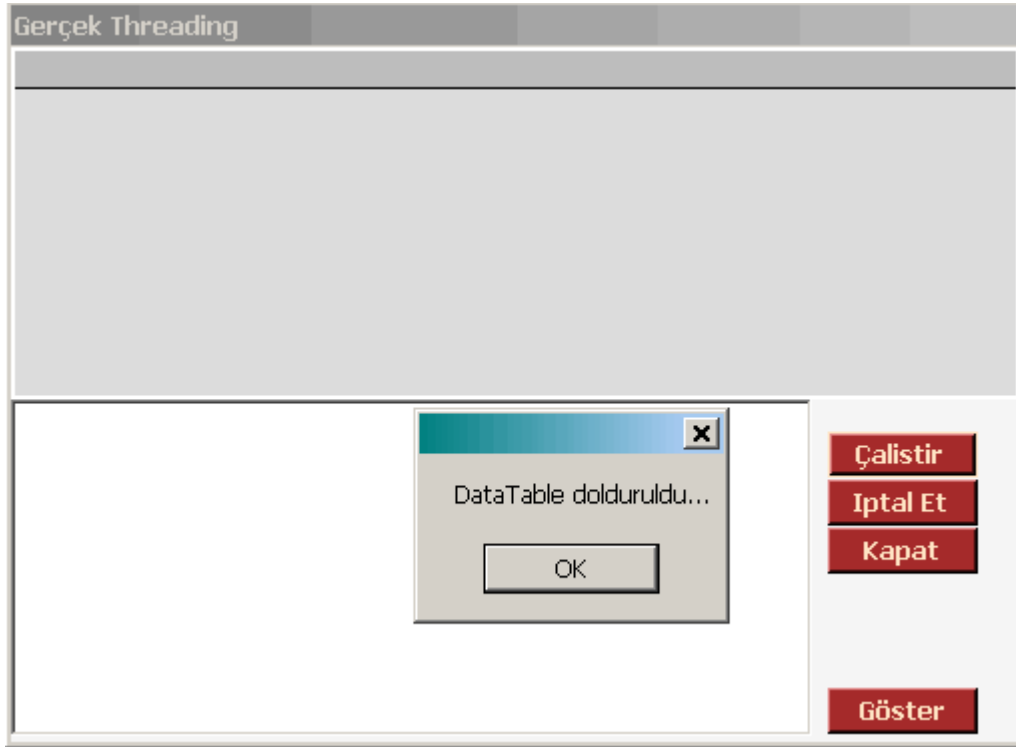


```

ds=new DataSet();
daNorthwind.Fill(ds);
conNorthwind.Close();
MessageBox.Show("DataTable dolduruldu...");
}
private void btnKapat_Click(object sender, System.EventArgs e)
{
    Close();
}
private void btnCalistir_Click(object sender, System.EventArgs e)
{
    Doldur();
}
private void btnGoster_Click(object sender, System.EventArgs e)
{
    Bagla();
}

```

Yazdığımız kodlar gayet basit. Sorgumuz bir SqlDataAdapter nesnesi ile, SqlConnection'ımız kullanılarak çalıştırılıyor ve daha sonra elde edilen veri kümesi DataSet'e aktarılıyor. Şimdi uygulamamızı bu haliyle çalıştıralım ve sorgumuzu Çalıştır başlıklı buton ile çalıştırdıktan sonra, textBox kontrolüne mouse ile tıklayıp bir şeyler yazmaya çalışalım.



Şekil 2. İş parçacığı olmadan programın çalışması.

Görüldüğü gibi sorgu sonucu elde edilen veri kümesi DataSet'e doldurulana kadar TextBox kontrolüne bir şey yazamadık. Çünkü işlemcimiz satır kodlarını işletmek ile meşguldü ve bizim TextBox kontrolümüze olan tıklamamızı ele almadı. Demekki buradaki sorgumuzu bir iş parçacığı içinde tanımlamalıyız. Nitekim programımız donmasın ve başka işlemlerde yapabilelim. Örneğin TextBox kontrolüne bir şeyler yazabilelim (bu noktada pek çok şey söylenebilir. Örneğin başka bir tablonun güncellenmesi gibi). Bu durumda yapmamız gereken kodlamayı inanıyorumki önceki makalelerden edindiğiniz bilgiler ile biliyorsunuzdur. Bu nedenle kodlarımızı detaylı bir şekilde açıklamadım. Şimdi gelin yeni kodlarımızı yazalım.

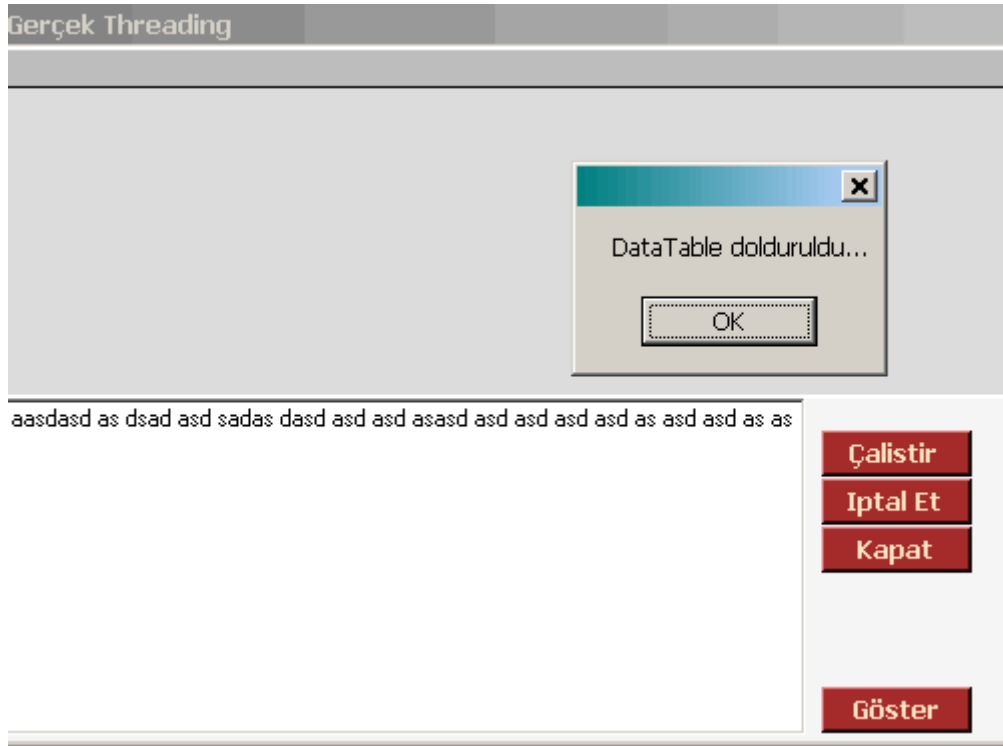
```
DataSet ds;
public void Bagla()
{
    if(!t1.IsAlive)
    {
        dataGrid1.DataSource=ds.Tables[0];
    }
}
public void Doldur()
{
    SqlConnection conNorthwind=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=sspi");
    conNorthwind.Open();
    SqlDataAdapter daNorthwind=new SqlDataAdapter("SELECT Products.* From [Order Details] Cross Join
Products",conNorthwind);
    ds=new DataSet();
    daNorthwind.Fill(ds);
    conNorthwind.Close();
    MessageBox.Show("DataTable dolduruldu...");
}
ThreadStart ts1;
Thread t1;
private void btnKapat_Click(object sender, System.EventArgs e)
{
    if(!t1.IsAlive)
    {
        Close();
    }
    else
    {
        MessageBox.Show("Is parçacigi henüz sonlandırılmadı...Daha sonra tekrar deneyin.");
    }
}
```

```

private void btnCalistir_Click(object sender, System.EventArgs e)
{
    ts1=new ThreadStart(Doldur);
    t1=new Thread(ts1);
    t1.Start();
}
private void btnIptalEt_Click(object sender, System.EventArgs e)
{
    t1.Abort();
}
private void btnGoster_Click(object sender, System.EventArgs e)
{
    Bagla();
}

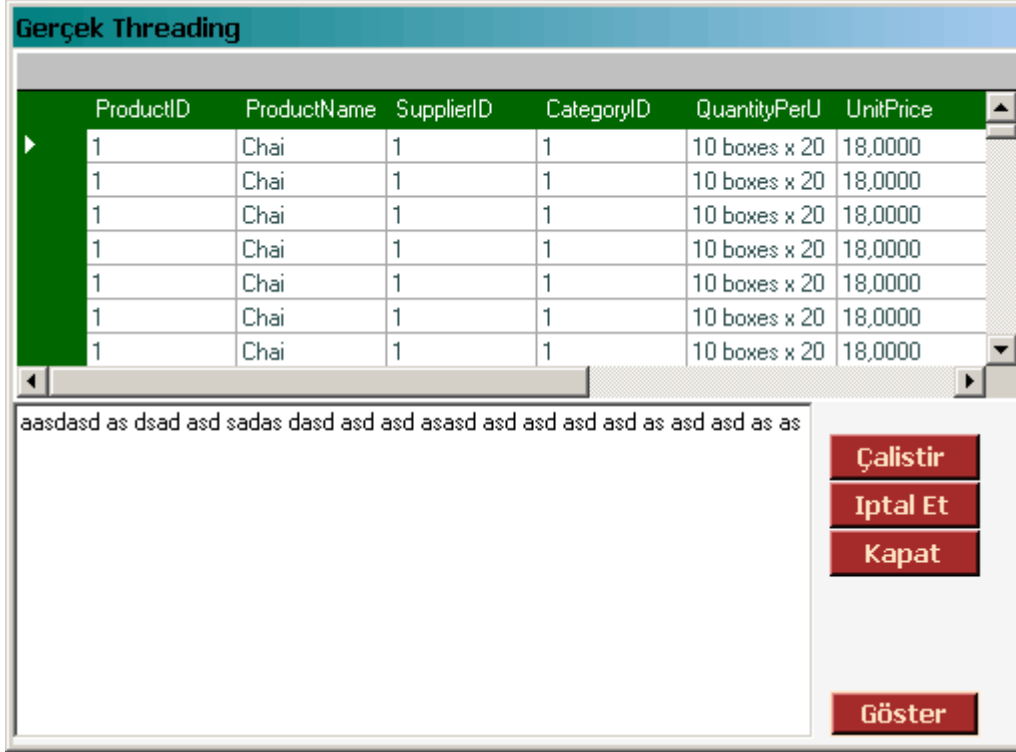
```

Şimdi programımızı çalıştıralım.



Şekil 3. İş Parçacığının sonucu.

Görüldüğü gibi bu yoğun sorgu çalışırken TextBox kontrolüne bir takım yazılar yazabildik. Üstelik programın çalışması hiç kesilmeden. Şimdi Göster başlıklı butona tıkladığımızda veri kümesinin DataGrid kontrolüne alındığını görürüz.



Şekil 4. Programın Çalışmasının Sonucu.

Geldik bir makalemizin daha sonuna. İlerliyen makalelerimizde Thred'leri daha derinlemesine incelemeye devam edeceğiz. Hepinize mutlu günler dilerim.

ArrayList Koleksiyonu ve DataGridView

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, veritabanlarındaki tablo yapısında olan bir ArrayList'i bir DataGridView kontrolüne nasıl veri kaynağı olarak bağlayacağımızı inceleyeceğiz. Bildiğiniz gibi ArrayList bir koleksiyon sınıfıdır ve System.Collections isim uzayında yer almaktadır. Genelde ArrayList koleksiyonlarını tercih etmemizin nedeni, dizilere olan üstünlüklerinden kaynaklanmaktadır.

En büyük tercih nedeni, normal dizilerin boyutlarının çalışma esnasında değiştirilemeyeşidir. Böyle bir işlemi gerçekleştirmek için, dizi elemanları yeni boyutlu başka boş bir diziye kopyalanır. Oysaki, ArrayList koleksiyonunda böyle bir durum söz konusu değildir. Koleksiyonu, aşağıdaki yapıcı metodu ile oluşturduğunuzda boyut belirtmezsiniz. Eleman ekledikçe, ArrayList'in kapasitesi otomatik olarak büyüyecektir.

```
public ArrayList();
```

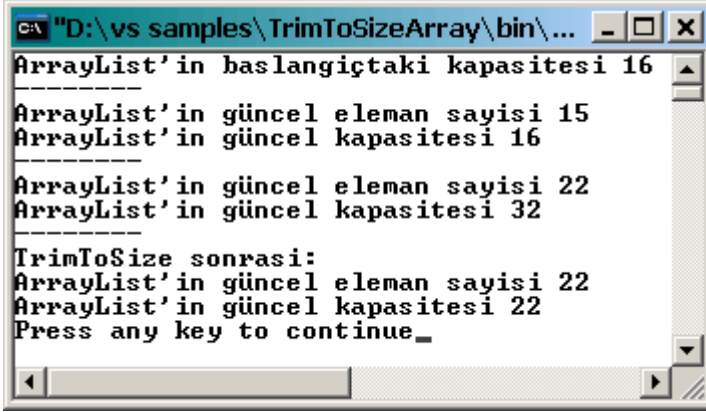
Bu şekilde tanımlanan bir ArrayList koleksiyonu varsayılan olarak 16 elemalı bir koleksiyon dizisi olur. Eğer kapasite aşılsa, koleksiyonun boyutu otomatik olarak artacaktır. Bu elbette karşımıza 17 elemanlı bir koleksiyonumuz varsa fazladan yer harcadığımız anlamında gelmektedir. Ancak sorunu TrimToSize metodu ile halledebiliriz. Dilerseniz bu konuyu aşağıdaki basit console uygulaması ile açıklayalım.

```

using System;
using System.Collections;
namespace TrimToArray
{
    class Class1
    {
        static void Main(string[] args)
        {
            ArrayList list=new ArrayList();
            Console.WriteLine("ArrayList'in başlangıçtaki kapasitesi "+list.Capacity.ToString()); /*Capacity koleksiyonun üst eleman
limitini verir. */
            Console.WriteLine("-----");
            for(int i=1;i<=15;++i)
            {
                list.Add(i); /* ArrayList koleksiyonunun sonuna eleman ekler. */
            }
            Console.WriteLine("ArrayList'in güncel eleman sayısı "+list.Count.ToString()); /* Count özelliği koleksiyonun o anki
eleman sayısını verir. */
            Console.WriteLine("ArrayList'in güncel kapasitesi "+list.Capacity.ToString());
            Console.WriteLine("-----");
            for(int j=1;j<8;++j)
            {
                list.Add(j);
            }
            Console.WriteLine("ArrayList'in güncel eleman sayısı "+list.Count.ToString());
            Console.WriteLine("ArrayList'in güncel kapasitesi "+list.Capacity.ToString());
            Console.WriteLine("-----");
            list.TrimToSize(); /* TrimToSize dizideki eleman sayısı ile kapasiteyi eşitler. */
            Console.WriteLine("TrimToSize sonrası:");
            Console.WriteLine("ArrayList'in güncel eleman sayısı "+list.Count.ToString());
            Console.WriteLine("ArrayList'in güncel kapasitesi "+list.Capacity.ToString());
        }
    }
}

```

Bu örneği çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 1. TrimToSize'in etkisi.

Görüldüğü gibi koleksiyonumuz ilk oluşturulduğunda kapasitesi 16'dır. Daha sonra koleksiyona 15 eleman ekledik. Halen koleksiyonun kapasite limitleri içinde olduğumuzdan, kapasitesi 16'dır. Ancak sonra 8 eleman daha ekledik. 17'nci eleman koleksiyona girdiğinde, koleksiyonun kapasitesi otomatik olarak iki katına çıkar. Bu durumda Capacity özelliğimiz 32 değerini verecektir. TrimToSize metodunu uyguladığımızda koleksiyonun kapasitesinin otomatik olarak eleman sayısı ile eşleştirildiğini görürüz.

Örneğimizde koleksiyonumuza eleman eklemek için Add metodunu kullandık. Add metodu her zaman yeni elemanı koleksiyonun sonuna ekler. Eğer koleksiyonda araya eleman eklemek istiyorsak insert metodunu, koleksiyonumuzdan bir eleman çıkartmak istediğimizde ise Remove metodunu kullanırız. Insert metodunun prototipi aşağıdaki gibidir.

```
public virtual void Insert(int index,object value );
```

İlk parametremiz 0 indeks tabanlı bir değerdir ve object tipindeki ikinci parametre değerinin hangi indeksli eleman olarak yerleştirileceğini belirtmektedir. Dolayısıyla bu elemanın insert edildiği yerdeki eleman bir ileriye ötelenmiş olacaktır. Remove metodu ise belirtilen elemanı koleksiyondan çıkartmaktadır. Prototipi aşağıdaki gibidir.

```
public virtual void Remove(object ob);
```

Metodumuz direkt olarak, çıkartılmak istenen elemanın değerini alır. ArrayList koleksiyonu, Remove metoduna alternatif başka metodlara sahiptir. Bunlar, RemoveAt ve RemoveRange metodlarıdır. RemoveAt metodu parametre olarak bir indeks değeri alır ve bu indeks değerindeki elemanı koleksiyondan çıkartır. Eğer girilen indeks değeri 0 dan küçük yada koleksiyonun eleman sayısına eşit veya büyük ise ArgumentOutOfRangeException istisnası fırlatılır.

RemoveRange metodu ise, ilk parametrede belirtilen indeks'ten, ikinci parametrede belirtilen sayıda elemanı koleksiyondan çıkartır. Elbette eğer indeks değeri 0 dan küçük yada koleksiyonun eleman sayısına eşit veya büyük ise ArgumentOutOfRangeException istisnası alınır. Tabi girdiğimiz ikinci parametre değeri, çıkartılmak istenen eleman sayısını, indeksten itibaren ele alındığında, koleksiyonun count özelliğinin değerinin üstüne çıkabilir. Bu durumda ise ArgumentException istisnası üretilecektir.

```
public virtual void RemoveAt(int index);
```

```
public virtual void RemoveRange(int index,int count);
```

Şimdi derseniz bu metodları küçük bir console uygulaması ile deneyelim.

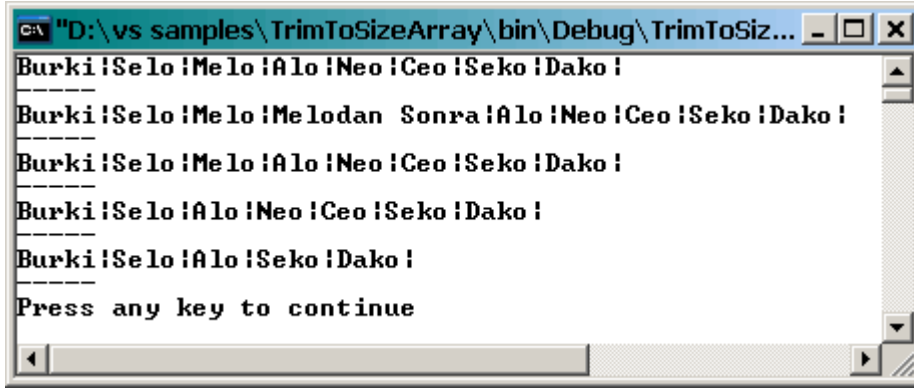
```
using System;
using System.Collections;
namespace TrimToSizeArray
{
    class Class1
    {
        static void Main(string[] args)
        {
            ArrayList kullanicilar=new ArrayList(); /* ArrayList koleksiyonumuz oluşturuluyor. */
            /* ArrayList koleksiyonumuza elemanlar ekleniyor. */
            kullanicilar.Add("Burki");
            kullanicilar.Add("Selo");
            kullanicilar.Add("Melo");
            kullanicilar.Add("Alo");
            kullanicilar.Add("Neo");
            kullanicilar.Add("Ceo");
            kullanicilar.Add("Seko");
            kullanicilar.Add("Dako");
            /* Foreach döngüsü yardımıyla, koleksiyonumuz içindeki tüm elemanları ekrana yazdırıyoruz. Elemanların herbirinin object
            tipinden ele alındığına dikkat edin. */
            foreach(object k in kullanicilar)
            {
                Console.Write(k.ToString()+"|");
            }
            Console.WriteLine();
            Console.WriteLine("-----");
            kullanicilar.Insert(3,"Melodan Sonra"); /* 3 noldu indeks'e "Melodan Sonra" elemanını yerleştirir. Bu durumda, "Alo" isimli
            eleman ve sonrakiler bir ileriye kayarlar. */
            foreach(object k in kullanicilar)
            {
                Console.Write(k.ToString()+"|");
            }
            Console.WriteLine();
            Console.WriteLine("-----");
            kullanicilar.Remove("Melodan Sonra"); /* "Melodan Sonra" isimli elemanı koleksiyondan çıkartır. */
            foreach(object k in kullanicilar)
            {
                Console.Write(k.ToString()+"|");
            }
            Console.WriteLine();
            Console.WriteLine("-----");
            kullanicilar.RemoveAt(2); /* 2nci indeks'te bulunan eleman koleksiyondan çıkartılır. Yani "Melo" çıkartılır. */
            foreach(object k in kullanicilar)
            {
                Console.Write(k.ToString()+"|");
            }
            Console.WriteLine();
        }
    }
}
```

```

        Console.WriteLine("-----");
        kullanicilar.RemoveRange(3,2); /* 3ncü indeks'ten itibaren, 2 eleman koleksiyondan çıkartılır. Yani "Neo" ve "Ceo"
        koleksiyondan çıkartılır. */
        foreach(object k in kullanicilar)
        {
            Console.Write(k.ToString()+"|");
        }
        Console.WriteLine();
        Console.WriteLine("-----");
    }
}
}

```

Uygulamamızı çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 2. Insert, Remove, RemoveAt ve RemoveRange metodları.

ArrayList koleksiyonu ile ilgili bu bilgilerden sonra sıra geldi DataGridView ile ilişkili olan kısma. Bildiğiniz gibi ArrayList'ler tüm koleksiyon sınıfları gibi elemanları object olarak tutarlar. Dolayısıyla bir sınıf nesnesinde bir ArrayList koleksiyonuna eleman olarak ekleyebiliriz. Şimdi geliştireceğimiz örnek uygulamada, bir veri tablosu gibi davranan bir ArrayList oluşturacağız. Bir veritablosu gibi alanları olacak. Peki bunu nasıl yapacağız?

Öncelikle, tablodaki her bir alanı birer özellik olarak tutacak bir sınıf tanımlayacağız. Bu durumda, bu sınıftan türettiğimiz her bir nesnede sahip olduğu özellik değerleri ile, tablodaki bir satırlık veriyi temsil etmiş olacak. Daha sonra bu nesneyi, oluşturduğumuz ArrayList koleksiyonuna ekleyeceğiz. Son olarakta bu ArrayList koleksiyonunu , DataGridView kontrolümüze veri kaynağı olarak bağlayacağız. Öncelikle Formumuzu tasarlayalım.

Şekil 3. Form Tasarımımız.

Şimdi koleksiyonumuzun taşıyacağı nesnelerin sınıfını tasarlayalım.

```
using System;

using System.Collections;

namespace ArrayListAndDataGrid
{
    public class MailList
    {
        public MailList(string k,string m)
        {
            mailAdresi=m;
            kullanici=k;
        }
        public MailList()
        {
        }
        protected string mailAdresi;
        protected string kullanici;
        public string MailAdresi
        {
            get
            {
```

```

        return mailAdresi;
    }
    set
    {
        mailAdresi=value;
    }
}
public string Kullanici
{
    get
    {
        return kullanici;
    }
    set
    {
        kullanici=value;
    }
}
}
}

```

MailList sınıfımız Kullanici ve MailAdresi isimli iki özelliğe sahip. İşte bunlar tablo alanlarımızı temsil etmektedir. Şimdi program kodlarımızı yazalım.

```

ArrayList mList; /* ArrayList koleksiyonumuzu tanımlıyoruz. */
private void Form1_Load(object sender, System.EventArgs e)
{
    mList=new ArrayList(); /* Form yüklenirken, ArrayList koleksiyonumuz oluşturuluyor. */
    lblElemanSayisi.Text=mList.Count.ToString(); /* Koleksiyonumuzun eleman sayısı alınıyor ve label kontrolüne yazdırılıyor. */
}
private void btnEkle_Click(object sender, System.EventArgs e)
{
    if((txtKullanici.Text.Length==0) && (txtMail.Text.Length==0))
    {
        MessageBox.Show("Lütfen veri girin");
    }
    else
    {
        MailList kisi=new MailList(txtKullanici.Text,txtMail.Text); /* MailList sınıfından bir nesne oluşturuluyor. Yapıcı metodumuz
parametre olarak, textBox kontrollerine girilen değerleri alıyor ve bunları ilgili alanlara atıyor. */
        mList.Add(kisi); /* MailList sınıfından oluşturduğumuz nesnemizi koleksiyonumuza ekliyoruz. */
        dgListe.DataSource=null; /* DataGrid' veri kaynağı olarak önce null değer atıyor. Nitekim, koleksiyonumuza her bir sınıf
nesnesi eklendiğinde, koleksiyon güncellenirken, dataGrid'in güncellenmediğini görürüz. Yapılacak tek şey veri kaynağını önce null
olarak ayarlamaktır. */
        dgListe.Refresh();
        dgListe.DataSource=mList; /* Şimdi işte, dataGrid kontrolümüze veri kaynağı olarak koleksiyonumuzu bağlıyoruz. */
        dgListe.Refresh();
        lblElemanSayisi.Text=mList.Count.ToString();
        txtKullanici.Clear();
    }
}

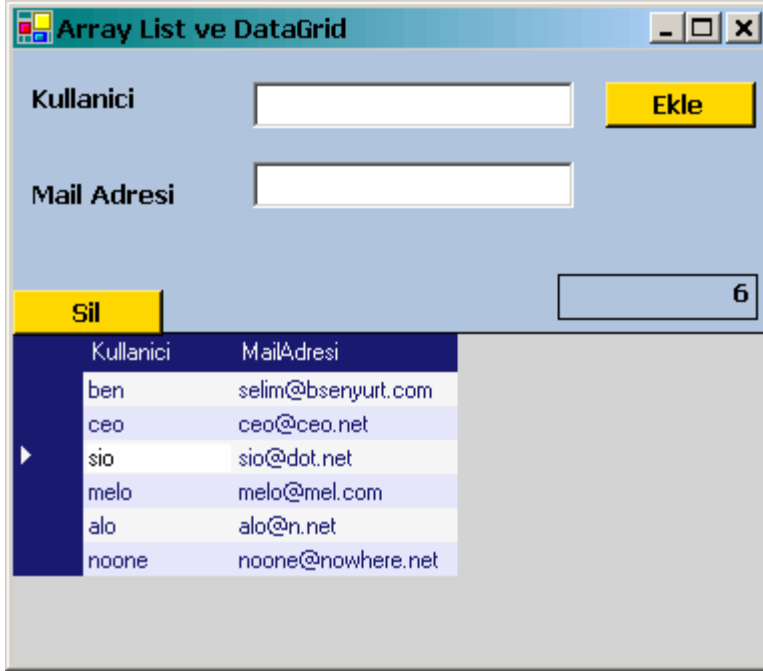
```

```

        txtMail.Clear();
    }
}
private void btnSil_Click(object sender, System.EventArgs e)
{
    int index;
    index = dgListe.CurrentCell.RowIndex; /* Silinmek istenen eleman DataGrid'te seçildiğinde, hangi satırın seçildiğini öğrenmek
    için CurrentCell.RowIndex özelliğini kullanıyoruz. */
    try
    {
        MailList kisi=new MailList(); /* Bir MailList nesnesi oluşturuyoruz. */
        kisi = (MailList)mList[index]; /* index değerimiz, DataGrid kontrolünde, bizim seçtiğimiz satırın, koleksiyonda karşılık gelen
        index değeridir. mList koleksiyonunda ilgili indeks'teki elemanı alıp (MailList) söz dizimi ile MailList tipinden bir nesneye
        dönüştürüyoruz. Nitekim koleksiyonlar elemanlarını object olarak tutarken, bu elemanları dışarı alırken açıkça bir dönüştürme işlemi
        uygulamamız gerekir. */
        mList.Remove(kisi); /* Remove metodu ile, kisi nesnemiz, dolayısıyla dataGrid kontrolünde seçtiğimiz eleman koleksiyondan
        çıkartılıyor. */
        dgListe.DataSource = null;
        dgListe.Refresh();
        dgListe.DataSource = mList;
        dgListe.Refresh();
        lblElemanSayisi.Text=mList.Count.ToString();
    }
    catch(Exception hata)
    {
        MessageBox.Show(hata.Message.ToString());
    }
}
}

```

Şimdi uygulamamızı çalıştıralım.



Şekil 4. Programın Çalışması

Bir sonraki adımımız bu ArrayList'in sahip olduğu verilerin gerçek bir tabloya yazdırılması olabilir. Bu adımın geliştirilmesini siz değerli Okurlarıma bırakıyorum. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Burak Selim ŞENYURT

selim@bsenyurt.com

Interface (Arayüz) Kullanımına Giriş

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, nesneye dayalı programlamanın önemli kavramlarından birisi olan arayüzleri incelemeye çalışacağız. Öncelikle, arayüz'ün tanımını yapalım. Bir arayüz, başka sınıflar için bir rehberdir. Bu kısa tanımın arkasında, deryalar gibi bir kavram denizi olduğunu söylemekte yarar buluyorum.. Arayüzün ne olduğunu tam olarak anlayabilmek için belkide asıl kullanım amacına bakmamız gerekmektedir.

C++ programlama dilinde, sınıflar arasında çok kalıtmılık söz konusu idi. Yani bir sınıf, birden fazla sınıftan türetilabiliyordu kalıtsal olarak. Ancak bu teknik bir süre sonra kodların dahada karmaşıklaşmasına ve anlaşılabilirliğin azalmasına neden oluyordu. Bu sebeben ötürü değerli Microsoft mimarları, C# dilinde, bir sınıfın sadece tek bir sınıfı kalıtsal olarak alabileceği kısıtlamasını getirdiler. Çok kalıtmılık görevini ise anlaşılması daha kolay arayüzlere bıraktılar. İşte arayüzleri kullanmamızın en büyük nedenlerinden birisi budur.

Diğer yandan, uygulamalarımızın geleceği açısından da arayüzlerin çok kullanışlı olabileceğini söylememiz gerekiyor. Düşününkü, bir ekip tarafından yazılan ve geliştirilen bir uygulamada görevlisiniz. Kullandığınız nesnelerin, türetildiği sınıflar zaman içerisinde,

gelişen yeniliklere adapte olabilmek amacıyla, sayısız yeni metoda, özelliğe vb.. sahip olduklarını farzedin. Bir süre sonra, nesnelerin türetildiği sınıflar içerisinde yer alan kavram kargaşasını, "bu neyi yapıyordu?, kime yapıyordu? , ne için yapıyordu?" gibi soruların ne kadar çok sorulduğunu düşünün. Oysa uygulamanızdaki sınıfların izleyeceği yolu gösteren rehber(ler) olsa fena mı olurdu? İşte size arayüzler. Bir arayüz oluşturun ve bu arayüzü uygulayan sınıfların hangi metodları, özellikleri vb kullanması gerektiğine karar verin. Programın gelişmesini gerekiyor?. Yeni niteliklere mi ihtiyacın var? İster kullanılan arayüzleri, birbirlerinden kalıtsal olarak türetin, ister yeni arayüzler tasarlayın. Tek yapacağınız sınıfların hangi arayüzlerini kullanacağını belirtmek olacaktır.

Bu açıklamalar ışığında bir arayüz nasıl tanımlanır ve hangi üyelere sahiptir bundan bahsedelim. Bir arayüz tanımlanması aşağıdaki gibi yapılır. Yazılan kod bloğunun bir arayüz olduğunu Interface anahtar sözcüğü belirtmektedir. Arayüz isminin başında I harfi kullanıldığına dikkat edin. Bu kullanılan sınıfın bir arayüz olduğunu anlamamıza yarayan bir isim kullanma tekniğidir. Bu sayede, sınıfların kalıtsal olarak aldığı elemanların arayüz olup olmadığını daha kolayca anlayabiliriz.

```
public interface IArayuz
```

```
{  
}
```

Tanımlama görüldüğü gibi son derece basit. Şimdi arayüzlerin üyelerine bir göz atalım. Arayüzler, sadece aşağıdaki üyelere sahip olabilirler.

Arayüz Üyeleri
özellikler (properties)
metodlar (methods)
olaylar (events)
indeksleyiciler (indexers)

Tablo 1. Arayüzlerin sahip olabileceği üyeler

Diğer yandan, arayüzler içerisinde aşağıdaki üyeler kesinlikle kullanılamazlar.

Arayüzlerde Kullanılamayan Üyeler
yapıcılar (constructors)

yokediciler (destructors)
alanlar (fields)

Tablo 2. Arayüzlerde kullanılmayan üyeler.

Arayüzler Tablo1 deki üyelere sahip olabilirler. Peki bu üyeler nasıl tanımlanır. Herşeyden önce arayüzler ile ilgili en önemli kural onun bir rehber olmasıdır. Yani arayüzler sadece, kendisini rehber alan sınıfların kullanacağı üyeleri tanımlarlar. Herhangibir kod satırı içermezler. Sadece özelliğin, metodun, olayın veya indeksleyicinin tanımı vardır. Onların kolay okunabilir olmalarını sağlayan ve çoklu kalıtım için tercih edilmelerine neden olan sebepte budur. Örneğin;

```
public interface IArayuz
{
    /* double tipte bir özellik tanımı. get ve set anahtar sözcüklerinin herhangi bir blok {} içermediğine dikkat edin. */
    double isim
    {
        get;
        set;
    }
    /* Yanlız okunabilir (ReadOnly) string tipte bir özellik tanımı. */
    string soyisim
    {
        get;
    }
    /* integer değer döndüren ve ili integer parametre alan bir metod tanımı. Metod tanımlarındada metodun dönüş tipi, parametreleri, ismi dışında herhangi bir kod satırı olmadığına dikkat edin. */
    int topla(int a,int b);
    /* Dönüş değeri olmayan ve herhangi bir parametre almayan bir metod tanımı. */
    void yaz();
    /* Bir indeksleyici tanımı */
    string this[int index]
    {
        get;
        set;
    }
}
```

Görüldüğü gibi sadece tanımlamalar mevcut. Herhangibir kod satırı mevcut değil. Bir arayüz tasarlarken uymamız gereken bir takım önemli kurallar vardır. Bu kurallar aşağıdaki tabloda kısaca listelenmiştir.

1	Bir arayüz'ün tüm üyeleri public kabul edilir. Private, Protected gibi belirteçler kullanamayız. Bunu yaptığımız takdirde örneğin bir elemanı private tanımladığımız takdirde, derleme zamanında aşağıdaki hatayı alırız. "The modifier 'private' is not valid for this item"
2	Diğer yandan bir metodu public olarak tanımlayamayız. Çünkü zaten varsayılan olarak bütün üyeler public tanımlanmış kabul edilir. Bir metodu public tanımladığımızda yine derleme zamanında aşağıdaki hatayı alırız. "The modifier 'public' is not valid for this item"
3	Bir arayüz, bir yapı(struct)'dan veya bir sınıf(class)'tan kalıtımla türetilemez. Ancak, bir arayüzü başka bir arayüzden veya arayüzlerden kalıtımsal olarak türetebiliriz.
4	Arayüz elemanlarını static olarak tanımlayamayız.
5	Arayüzlerin uygulandığı sınıflar, arayüzde tanımlanan bütün üyeleri kullanmak zorundadır.

Tablo 3. Uyulması gereken kurallar.

Şimdi bu kadar laftan sonra konuyu daha iyi anlayabilmek için basit ama açıklayıcı bir örnek geliştirelim. Önce arayüzümüzü tasarlayalım.

```
public interface IArayuz
{
    void EkranaYaz();
    int Yas
    {
        get;
        set;
    }
    string isim
    {
        get;
        set;
    }
}
```

Şimdide bu arayüzü kullanacak sınıfımızı tasarlayalım.

```
public class Kisiler:IArayuz
```

```
{
}
```

Şimdi bu anda uygulamayı derlersek, IArayuz'ündeki elemanları sınıfımız içinde kullanmadığımızdan dolayı aşağıdaki derleme zamanı hatalarını alırız.

'Interfaces1.Kisiler' does not implement interface member 'Interfaces1.IArayuz.EkranaYaz()'

'Interfaces1.Kisiler' does not implement interface member 'Interfaces1.IArayuz.isim'

'Interfaces1.Kisiler' does not implement interface member 'Interfaces1.IArayuz.Yas'

Görüldüğü gibi kullanmadığımız tüm arayüz üyeleri için bir hata mesajı oluştu. Bu noktada şunu tekrar hatırlatmak istiyorum,

Arayüzlerin uygulandığı sınıflar, arayüzde(lerde) tanımlanan tüm üyeleri kullanmak yani kodlamak zorundadır.

Şimdi sınıfımızı düzgün bir şekilde geliştirelim.

```
public class Kisiler:IArayuz /* Sınıfın kullanacağı arayüz burada belirtiliyor.*/
```

```
{
```

```
    private int y;
```

```
    private string i;
```

```
    /* Bir sınıfa bir arayüz uygulamamız, bu sınıfa başka üyeler eklememizi engellemez. Burada örneğin sınıfın yapıcı metodlarımızda düzenledik. */
```

```
    public Kisiler()
```

```
    {
```

```
        y=18;
```

```
        i="Yok";
```

```
    }
```

```
    /* Dikkat ederseniz özelliğin herşeyi, arayüzdeki ile aynı olmalıdır. Veri tipi, ismi vb... Bu tüm diğer arayüz üyelerinin, sınıf içerisinde uygulanmasında da geçerlidir. */
```

```
    public Kisiler(string ad,int yas)
```

```
    {
```

```
        y=yas;
```

```
        i=ad;
```

```
    }
```

```
    public int Yas
```

```
    {
```

```
        get
```

```
        {
```



```
        return y;
    }
    set
    {
        y=value;
    }
}
public string Isim
{
    get
    {
        return i;
    }
    set
    {
        i=value;
    }
}
public void EkranaYaz()
{
    Console.WriteLine("Adım:"+i);
    Console.WriteLine("Yaşım:"+y);
}
}
```

Şimdi oluşturduğumuz bu sınıfı nasıl kullanacağımıza bakalım.

```
class Class1
{
    static void Main(string[] args)
    {
        Kisiler kisi=new Kisiler("Burak",27);

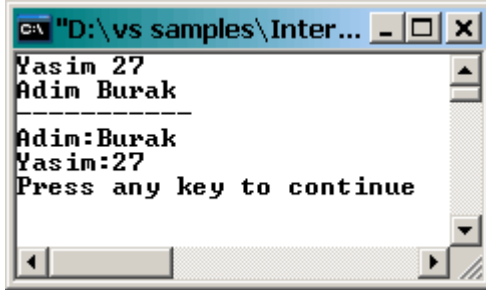
        Console.WriteLine("Yaşım "+kisi.Yas.ToString());
        Console.WriteLine("Adım "+kisi.Isim);
        Console.WriteLine("-----");
        kisi.EkranaYaz();
    }
}
```

```

    }
}

```

Uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 1. Uygulamanın Çalışması Sonucu.

Geldik bir makalemizin daha sonuna. Bu makalemizde arayüzlere kısa bir giriş yaptık. Bir sonraki makalemizde, bir sınıfa birden fazla arayüzün nasıl uygulanacağını incelemeye çalışacağız. Hepinize mutlu günler dilerim.

Arayüz(Interface), Sınıf(Class) ve Çoklu Kalıtım

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, arayüzleri incelemeye devam edeceğiz. Bir önceki makalemizde, arayüzleri kullanmanın en büyük nedenlerinden birisinin sınıflara çoklu kalıtım desteği vermesi olduğunu söylemiştik. Önce basit bir uygulama ile bunu gösterelim.

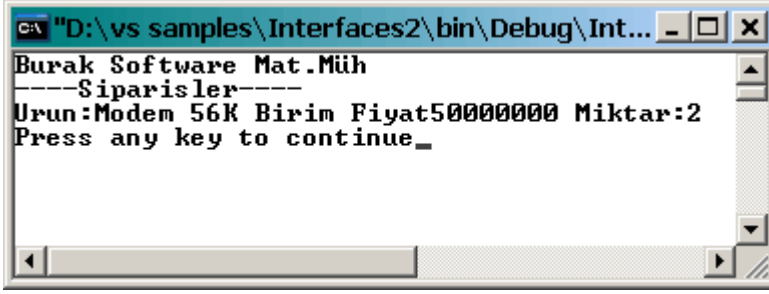
```

using System;
namespace Interfaces2
{
    public interface IMusteri
    {
        void MusteriDetay();
        int ID{get;}
        string Isim{get;set;}
        string Soyisim{get;set;}
        string Meslek{get;set;}
    }
    public interface ISiparis
    {
        int SiparisID{get;}
        string Urun{get;set;}
        double BirimFiyat{get;set;}
        int Miktar{get;set;}
    }
}

```

```
void SiparisDetay();
}
public class Sepet:IMusteri,ISiparis /* Sepet isimli sınıfımız hem IMusteri arayüzünü hemde ISiparis arayüzünü uygulayacaktır.
*/
{
    private int id,sipId,mkt;
    private string ad,soy,mes,ur;
    private double bf;
    public int ID
    {
        get{return id;}
    }
    public string Isim
    {
        get{return ad;}
        set{ad=value;}
    }
    public string Soyisim
    {
        get{return soy;}
        set{soy=value;}
    }
    public string Meslek
    {
        get{return mes;}
        set{mes=value;}
    }
    public void MusteriDetay()
    {
        Console.WriteLine(ad+" "+soy+" "+mes);
    }
    public int SiparisID
    {
        get{return sipId;}
    }
    public string Urun
    {
        get{return ur;}
        set{ur=value;}
    }
}
```

```
    }  
    public double BirimFiyat  
    {  
        get{return bf;}  
        set{bf=value;}  
    }  
    public int Miktar  
    {  
        get{return mkt;}  
        set{mkt=value;}  
    }  
    public void SiparisDetay()  
    {  
        Console.WriteLine("----Siparisler----");  
        Console.WriteLine("Urun:"+ur+" Birim Fiyat"+bf.ToString()+" Miktar:"+mkt.ToString());  
    }  
}  
class Class1  
{  
    static void Main(string[] args)  
    {  
        Sepet spt1=new Sepet();  
        spt1.Isim="Burak";  
        spt1.Soyisim="Software";  
        spt1.Meslek="Mat.Müh";  
        spt1.Urun="Modem 56K";  
        spt1.BirimFiyat=50000000;  
        spt1.Miktar=2;  
        spt1.MusteriDetay();  
        spt1.SiparisDetay();  
    }  
}
```



Şekil 1. Programın Çalışmasının Sonucu.

Yukarıdaki kodlarda aslında değişik olarak yaptığımız bir şey yok. Sadece oluşturduğumuz arayüzleri bir sınıfa uyguladık ve çok kalıtmıllığı gerçekleştirmiş olduk. Ancak bu noktada dikkat etmemiz gereken bir unsur vardır. Eğer arayüzler aynı isimli metodlara sahip olurlarsa ne olur? Bu durumda arayüzlerin uygulandığı sınıfta, ilgili metodu bir kez yazmamız yeterli olacaktır. Söz gelimi, yukarıdaki örneğimizde, Baslat isimli ortak bir metodun arayüzlerin ikisi içinde tanımlanmış olduğunu varsayalım.

```
public interface IMusteri
{
    void MusteriDetay();
    int ID{get;}
    string Isim{get;set;}
    string Soyisim{get;set;}
    string Meslek{get;set;}
    void Baslat();
}

public interface ISiparis
{
    int SiparisID{get;}
    string Urun{get;set;}
    double BirimFiyat{get;set;}
    int Miktar{get;set;}
    void SiparisDetay();
    void Baslat();
}
```

Şimdi bu iki arayüzde aynı metod tanımına sahip. Sınıfımızda bu metodları iki kez yazmak anlamsız olacaktır. O nedenle sınıfımızda aşağıdaki gibi tek bir Baslat metodu ekleriz. Sınıf nesnemizi oluşturduğumuzda, Baslat isimli metodu aşağıdaki gibi çalıştırabiliriz.

```
spt1.Baslat();
```

Fakat bazı durumlarda, arayüzlerdeki metodlar aynı isimlide olsalar, arayüzlerin uygulandığı sınıf içerisinde söz konusu metod, arayüzlerin her biri için ayrı ayrı yazılmak istenebilir. Böyle bir durumda ise sınıf içerisindeki metod yazımlarında arayüz isimlerini de belirtiriz.Örneğin;

```

void IMusteri.Baslat()
{
    Console.WriteLine("Müşteriler hazırlandı...");
}
void ISiparis.Baslat()
{
    Console.WriteLine("Siparişler hazırlandı...");
}

```

Metodların isimleri başında hangi arayüz için yazıldıklarına dikkat edelim. Diğer önemli bir nokta public belirtecinin kullanılmayıdır. Public belirtecini kullanmamız durumunda, "The modifier 'public' is not valid for this item" derleyici hatasını alırız. Çünkü, metodumuzu public olarak tanımlamaya gerek yoktur. Nitekim, bu metodların kullanıldığı sınıflara ait nesnelerden, bu metodları çağırmak istediğimizde doğrudan çağıramadığımız görürüz. Çünkü derleyici hangi arayüzde tanımlanmış metodun çağırılması gerektiğini bilemez. Bu metodları kullanabilmek için, nesne örneğini ilgili arayüz tiplerine dönüştürmemiz gerekmektedir. Bu dönüştürmenin yapılması ilgili sınıf nesnesinin, arayüz tipinden değişkenlere açık bir şekilde dönüştürülmesi ile oluşur. İşte bu yüzden bu tip metodlar, tanımlandıkları sınıf içinde public yapılamazlar. Bu açıkça dönüştürme işlemide aşağıdaki örnek satırlarda görüldüğü gibi olur.

```

IMusteri m=(IMusteri)spt1;
ISiparis s=(ISiparis)spt1;

```

İşte şimdi istediğimiz metodu, bu değişken isimleri ile birlikte aşağıdaki örnek satırlarda olduğu gibi çağırabiliriz.

```

m.Baslat();
s.Baslat();

```



Şekil 2. Programın Çalışmasının Sonucu.

Geldik bir makalemizin daha sonuna ilerleyen makalelerimizde arayüzleri incelemeye devam edeceğiz. Hepinize mutlu günler dilerim.

Arayüzler'de is ve as Anahtar Sözcüklerinin Kullanımı

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, arayüzlerde **is** ve **as** anahtar kelimelerinin kullanımını inceleyeceğiz. Bir sınıfa arayüz(ler) uyguladığımızda, bu arayüzlerde tanımlanmış metodları çağırmak için çoğunlukla tercih edilen bir teknik vardır. O da, bu sınıfa ait nesne örneğini, çalıştırılacak metodun tanımlandığı arayüz tipine dönüştürmek ve bu şekilde çağırmaktır. Bu teknik, her şeyden önce, program kodlarının okunabilirliğini ve anlaşılabilirliğini arttırmaktadır. Öyleki, bir isim uzayında yer alan çok sayıda arayüzün ve sınıfın yer aldığı uygulamalarda bu tekniği uygulayarak, hangi arayüze ait metodun çalıştırıldığı daha kolay bir şekilde gözlemlenebilmektedir. Diğer yandan bu teknik, aynı metod tanımlamalarına sahip arayüzler için de kullanılır ki bunu bir önceki makalemizde işlemiştik.

Bu teknik ile ilgili olarak, dikkat edilmesi gereken bir noktada vardır. Bir sınıfa ait nesne örneğini, bu sınıfa **uygulamadığımız** bir arayüze ait herhangi bir metodu çalıştırmak için, ilgili arayüz tipine dönüştürdüğümüzde **InvalidCastException** istisnasını alırız. Bu noktayı daha iyi vurgulayabilmek için aşağıdaki örneğimizi göz önüne alalım. Bu örnekte iki arayüz yer almakta olup, tanımladığımız sınıf, bu arayüzlerden sadece bir tanesini uygulamıştır. Ana sınıfımızda, bu sınıfa ait nesne örneği, uygulanmamış arayüz tipine dönüştürülmüş ve bu arayüzdeki bir metod çağırılmak istenmiştir.

```
using System;
namespace Interface3
{
    public interface IKullanilmayan
    {
        void Yaz();
        void Bul();
    }
    public interface IKullanilan
    {
        void ArayuzAdi();
    }
    public class ASinifi:IKullanilan
    {
        public void ArayuzAdi()
        {
            Console.WriteLine("Arayüz adl:IKullanilan");
        }
    }
    class Class1
    {
        static void Main(string[] args)
        {
            ASinifi a=new ASinifi();
            IKullanilan Kul=(IKullanilan)a;
            Kul.ArayuzAdi();
            IKullanilmayan anKul=(IKullanilmayan)a;
```

```

        anKul.Yaz();
    }
}

```

Bu örneği derlediğinizde herhangi bir derleyici hatası ile karşılaşmazsınız. Ancak çalışma zamanında

"System.InvalidCastException: Specified Cast Is Invalid" çalışma zamanı hatasını alırsınız. İşte bu sorunu **is** veya **as** anahtar sözcüklerinin kullanıldığı iki farklı teknikten birisi ile çözebiliriz. **Is** ve **as** bu sorunun çözümünde aynı amaca hizmet etmekle beraber aralarında önemli iki fark vardır.

Is anahtar kelimesi aşağıdaki formasyonda kullanılır.

nesne is tip

Is anahtar kelimesi nesne ile tipi karşılaştırır. Yani belirtilen nesne ile, bir sınıfı veya arayüzü kıyaslarlar. Bu söz dizimi bir if karşılaştırmasında kullanılır ve eğer nesnenin üretildiği sınıf, belirtilen tip'teki arayüzden uygulanmışsa bu koşullu ifade true değerini döndürecek. Aksi durumda false değerini döndürür. Şimdi bu tekniği yukarıdaki örneğimize uygulayalım. Yapmamız gereken değişiklik Main metodunda yer almaktadır.

```

static void Main(string[] args)
{
    ASinifi a=new ASinifi();
    IKullanilan Kul=(IKullanilan)a;
    Kul.ArayuzAdi();
    if(a is IKullanilmayan)
    {
        IKullanilmayan anKul=(IKullanilmayan)a;
        anKul.Yaz();
    }
    else
    {
        Console.WriteLine("ASinifi, IKullanilmayan arayüzünü uygulamamıştır.");
    }
}

```



Şekil 1: is Anahtar Kelimesinin Kullanımı.

If koşullu ifadesinde, a isimli nesneyi oluşturduğumuz ASinifi sınıfına, IKullanilmayan arayüzünü uygulayıp uyguladığımızı kontrol etmekteyiz. Sonuç false değerini döndürecektir. Nitekim, ASinifi sınıfına, IKullanilmayan arayüzünü uygulamadık.

Is anahtar sözcüğü arayüzler dışında sınıflar içinde kullanabiliriz. Bununla birlikte **is** anahtar sözcüğünü kullanıldığında, program kodu Intermediate Language (IL) çevrildiğinde, yapılan denetlemenin iki kere tekrar edildiğini görürüz. Bu verimliliği düşürücü bir etkidir. İşte **is** yerine **as** anahtar sözcüğünü tercih etmemizin nedenlerinden biriside budur. Diğer taraftan **is** ve **as** teknikleri, döndürdükleri değerler bakımından da farklılık gösterir. **Is** anahtar kelimesi, bool tipinde true veya false değerlerini döndürür. **As** anahtar kelimesi ise, bir nesneyi, bu nesne sınıfına uygulanmış bir arayüz tipine dönüştürür. Eğer nesne sınıfı, belirtilen arayüzü uygulamamışsa, dönüştürme işlemi yinede yapılır, fakat dönüştürülmenin aktarıldığı değişken null değerine sahip olur. **As** anahtar kelimesinin formu aşağıdaki gibidir.

sınıf nesne1=nesne2 as tip

Burada eğer nesneye belirtilen tipi temsil eden arayüz uygulanmamışsa, nesne null değerini alır. Aksi durumda nesne belirtilen tipe dönüştürülür. İşte **is** ile **as** arasındaki ikinci farkta budur. Konuyu daha iyi anlayabilmek için **as** anahtar kelimesini yukarıdaki örneğimize uygulayalım.

```
static void Main(string[] args)
{
    ASinifi a=new ASinifi();
    IKullanilan Kul=(IKullanilan)a;
    Kul.ArayuzAdi();
    IKullanilmayan anKul=a as IKullanilmayan;
    if(anKul!=null)
    {
        anKul.Yaz();
    }
    else
    {
        Console.WriteLine("ASinifi IKullanilmayan tipine dönüştürülemedi");
    }
}
```

Burada a nesnemiz ASinifi sınıfının örneğidir. **As** ile bu örneği IKullanilmayan arayüzü tipinden anKul değişkenine aktarmaya çalışıyoruz. İşte bu noktada, ASinifi, IKullanilmayan arayüzünü uygulamadığı için, anKul değişkeni null değerini alacaktır. Sonra if koşullu ifadesi ile, anKul 'un null olup olmadığı kontrol ediyoruz. Uygulamayı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 2: as Anahtar Kelimesinin Kullanımı

Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Bir Arayüz, Bir Sınıf ve Bir Tablo

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, bir arayüzü uygulayan sınıf nesnelerinden faydalanarak, bir Sql tablosundan nasıl veri okuyacağımızı ve değişiklikleri veritabanına nasıl göndereceğimizi incelemeye çalışacağız. Geliştireceğimiz örnek, arayüzlerin nasıl oluşturulduğu ve bir sınıfa nasıl uygulandığını incelemekle yetinmeyecek, Sql veritabanımızdaki bir tablodaki belli bir kayda ait verilerin bu sınıf nesnelerine nasıl aktarılacağını da işleyecek. Kısacası uygulamamız, hem arayüzlerin hem sınıfların hemde Sql nesnelerinin kısa bir tekrarı olacak.

Öncelikle uygulamamızın amacından bahsedelim. Uygulamamızı bir Windows uygulaması şeklinde geliştireceğiz. Kullanacağımız veri tablosunda arkadaşlarımızla ilgili bir kaç veriyi tutuyor olacağız. Kullanıcı, Windows formunda, bu tablodaki alanlar için Primary Key niteliği taşıyan bir ID değerini girerek, buna karşılık gelen tablo satırına ait verilerini elde edicek. İsteddiği değişiklikleri yaptıktan sonra ise bu değişiklikleri tekrar veritabanına gönderecek. Burada kullanacağımız teknik makalemizin esas amacı olacak. Bu kez veri tablosundan çekip aldığımız veri satırının programdaki eşdeğeri, oluşturacağımız sınıf nesnesi olacak. Bu sınıfımız ise, yazmış olduğumuz arayüzü uygulayan bir sınıf olacak. Veriler sınıf nesnesine, satırdaki her bir alan değeri, aynı isimli özelliğe denk gelecek şekilde yüklenecek. Yapılan değişiklikler yine bu sınıf nesnesinin özelliklerinin sahip olduğu değerlerin veri tablosuna gönderilmesi ile gerçekleştirilecek.

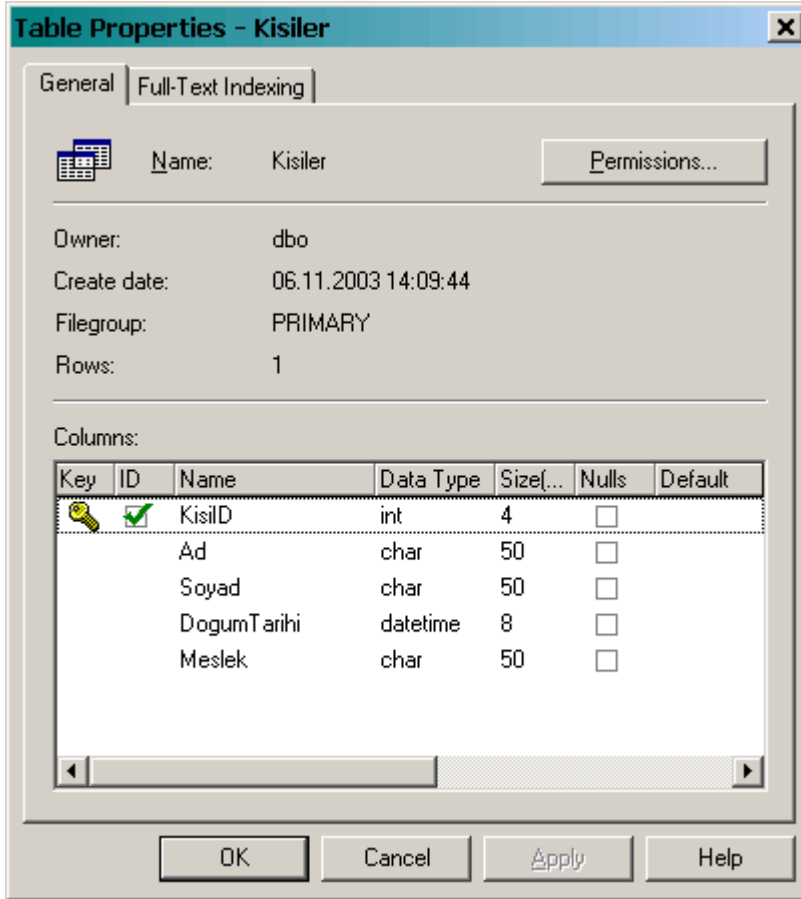
Uygulamamızda, verileri Sql veritabanından çekmek için, SqlConnection isim uzayında yer alan SqlConnection ve SqlDataReader nesnelerini kullanacağız. Hatırlayacağınız gibi SqlConnection nesnesi ile , bağlanmak istediğimiz veritabanına, bu veritabanının bulunduğu sunucu üzerinden bir bağlantı tanımlıyoruz. SqlDataReader nesnemiz ile de, sadece ileri yönlü ve yalnız okunabilir bir veri akımı sağlayarak, aradığımız kayda ait verilerin elde edilmesini sağlıyoruz.

Şimdi uygulamamızı geliştirmeye başlayalım. Öncelikle vs.net ortamında bir Windows Application oluşturalım. Burada aşağıdaki gibi bir form tasarlayalım.



Şekil 1. Form tasarımı.

Kullanıcı bilgilerini edinmek istediği kişinin ID'nosunu girdikten sonra, Getir başlıklı butona tıklayarak ilgili satırın tüm alanlarına ait verileri getirecek. Ayrıca, kullanıcı veriler üzerinde değişiklik yapabilecek ve bunlarda Güncelle başlıklı butona tıklayarak Sql veritabanındaki tablomuza aktarabilecek. Sql veritabanında yer alan Kisiler isimli tablomuzun yapısı aşağıdaki gibidir.



Şekil 2. Tablomuzun yapısı.

Şimdi gelelim işin en önemli ve anahtar kısımlarına. Program kodlarımız. Öncelikle arayüzümüzü tasarlayalım. Arayüzümüz, sonra oluşturacağımız sınıf için bir rehber olacak. Sınıfımız, veri tablomuzdaki alanları birer özellik olarak taşıyacağına göre arayüzümüzde bu özellik tanımlarının yer alması gerektiğini söyleyebiliriz. Ayrıca ilgili kişiye ait verileri getirecek bir metodumuzda olmalıdır. Elbette bu arayüze başka amaçlar için üye tanımlamalarıda ekleyebiliriz. Bu konuda tek sınır bizim hayal gücümüz. İşin gerçeği bu makalemizde hayal gücümü biraz kısım konunun daha fazla dağılmaması amacıyla :) . (Ama siz, örneğin kullanıcının yeni girdiği verileri veritabanına yazacak bir metod tanımında bir üye olarak ekleyebilir ve gerekli kodlamaları yapabilirsiniz.) İşte arayüzümüzün kodları.

```
public interface IKisi
```

```
{
```

```
    /* Öncelikle tablomuzdaki her alana karşılık gelen özellikler için tanımlamalarımızı yapıyoruz.*/
```

```
    int KisiID /* KisiID, tablomuzda otomatik artan ve primary key olan bir alandır. Dolayısıyla programcının var olan bir KisiID'sini
```

değiştirmemesi gerekir. Bu nedenle sadece okunabilir bir özellik olarak tanımlanmasına izin veriyoruz. */

```
{
    get;
}
string Ad /* Tablomuzdaki char tipindeki Ad alanımız için string tipte bir alan.*/
{
    get;set;
}
string Soyad
{
    get;set;
}
DateTime DogumTarihi /* Tablomuzda, DogumTarihi alanımız datetime tipinde olduğundan, DateTime tipinde bir özellik tanımlanmasına izin veriyoruz.*/
{
    get;set;
}
string Meslek
{
    get;set;
}
void Bul(int KID); /* Bul metod, KID parametresine göre, tablodan ilgili satıra ait verileri alacak ve alanlara karşılık gelen özelliklere atayacak metodumuzdur.*/
}
```

Şimdide bu arayüzümüzü uygulayacağımız sınıfımızı oluşturalım. Sınıfımız IKisi arayüzünde tanımlanan her üyeyi uygulamak zorundadır. Bu bildiğiniz gibi arayüzlerin bir özelliğidir.

```
public class CKisi:IKisi /* IKisi arayüzünü uyguluyoruz.*/
{
    /* Öncelikle sınıftaki özelliklerimiz için, verilerin tutulacağı alanları tanımlıyoruz.*/
    private int kisiID;
    private string ad;
    private string soyad;
    private DateTime dogumTarihi;
    private string meslek;
    /* Arayüzümüzde yer alan üyeleri uygulamaya başlıyoruz.*/
    public int KisiID
    {
        get{ return kisiID;}
    }
}
```

```

    }
    public string Ad
    {
        get{return ad;}set{ad=value;}
    }
    public string Soyad
    {
        get{return soyad;}set{soyad=value;}
    }
    public DateTime DogumTarihi
    {
        get{return dogumTarihi;}set{dogumTarihi=value;}
    }
    public string Meslek
    {
        get{return meslek;}set{meslek=value;}
    }
    public void Bul(int KID)
    {
        /* Öncelikle Sql Veritabanımıza bir bağlantı açıyoruz.*/
        SqlConnection conFriends=new SqlConnection("data source=localhost;integrated security=sspi;initial catalog=Friends");
        /* Tablomuzdan, kullanıcının bu metoda parametre olarak gönderdiği KID değerini baz alarak, ilgili KisiID'ye ait verileri elde
        edecek sql kodunu yazıyoruz.*/
        string sorgu="Select * From Kisiler Where KisiID="+KID.ToString();
        /* SqlCommand nesnemiz yardımıyla sql sorgumuzu çalıştırılmak üzere hazırlıyoruz.*/
        SqlCommand cmd=new SqlCommand(sorgu,conFriends);
        SqlDataReader rd;/* SqlDataReader nesnemizi yaratıyoruz.*/
        conFriends.Open(); /* Bağlantımızı açıyoruz. */
        rd=cmd.ExecuteReader(CommandBehavior.CloseConnection); /* ExecuteReader ile sql sorgumuzu çalıştırıyoruz ve sonuç
        kümesi ile SqlDataReader nesnemiz arasında bir akım(stream) açıyoruz. CommandBehavior.CloseConnection sayesinde,
        SqlDataReader nesnemizi kapattığımızda, SqlConnection nesnemizinde otomatik olarak kapanmasını sağlıyoruz.*/
        while(rd.Read())
        {
            /* Eğer ilgili KisiID'ye ait bir veri satırı bulunursa, SqlDataReader nesnemizin Read metodu sayesinde, bu satıra ait
            verileri sınıfımızın ilgili alanlarına aktarıyoruz. Böylece, bu alanların atandığı sınıf özellikleride bu veriler ile dolmuş oluyor.*/
            kisiID=(int)rd["KisiID"];
            ad=rd["Ad"].ToString();
            soyad=rd["Soyad"].ToString();
            dogumTarihi=(DateTime)rd["DogumTarihi"];

```

```

        meslek=rd["Meslek"].ToString();
    }
    rd.Close();
}
public CKisi()
{
}
}

```

Artık IKisi arayüzünü uygulayan, CKisi isimli bir sınıfımız var.Şimdi Formumuzun kodlarını yazmaya başlayabiliriz. Öncelikle module düzeyinde bir CKisi sınıf nesnesi tanımlayalım.

```
CKisi kisi=new CKisi();
```

Bu nesnemiz veri tablosundan çektiğimiz veri satırına ait verileri taşıyacak. Kullanıcı Getir başlıklı button kontrolüne bastığında olucak olayları gerçekleştirecek kodları yazalım.

```

private void btnGetir_Click(object sender, System.EventArgs e)
{
    int id=Convert.ToInt32(txtKisiID.Text.ToString()); /* Kullanıcının TextBox kontrolüne girdiği ID değeri Convert sınıfının ToInt32
    metodu ile Integer'a çeviriyoruz.*/
    kisi.Bul(id); /* Kisi isimli CKisi sınıfından nesne örneğimizin Bul metodunu çağırıyoruz.*/
    Doldur(); /* Doldur Metodu, kisi nesnesinin özellik değerlerini, Formumuzdaki ilgili kontrollere alarak, bir nevi veri bağlama
    işlemini gerçekleştirmiş oluyor.*/
}

```

Şimdide Doldur metodumuzun kodlarını yazalım.

```

public void Doldur()
{
    txtAd.Text=kisi.Ad.ToString(); /* txtAd kontrolüne, kisi nesnemizin Ad özelliğinin şu anki değeri yükleniyor. Yani ilgili veri
    satırının ilgili alanı bu kontrole bağlanmış oluyor.*/
    txtSoyad.Text=kisi.Soyad.ToString();
    txtMeslek.Text=kisi.Meslek.ToString();
    txtDogumTarihi.Text=kisi.DogumTarihi.ToShortDateString();
    lblKisiID.Text=kisi.KisiID.ToString();
}

```

Evet görüldüğü gibi artık aradığımız kişiye ait verileri formumuzdaki kontrollere yükleyebiliyoruz. Şimdi TextBox kontrollerimizin TextChanged olaylarını kodlayacağız. Burada amacımız, TextBox'larda meydana gelen değişikliklerin anında, CKisi sınıfından türettiğimiz Kisi nesnesinin ilgili özelliklerine yansıtılabilmesi. Böylece yapılan değişiklikler anında nesnemize yansıtılacak. Bu nedenle aşağıdaki kodları ekliyoruz.

/* Metodumuz bir switch case ifadesi ile, aldığı ozellikAdi parametresine göre, CKisi isimli sınıfımıza ait Kisi nesne örneğinin ilgili özelliklerini değiştiriyor.*/

```
public void Degistir(string ozellikAdi,string veri)
{
    switch(ozellikAdi)
    {
        case "Ad":
        {
            kisi.Ad=veri;
            break;
        }
        case "Soyad":
        {
            kisi.Soyad=veri;
            break;
        }
        case "Meslek":
        {
            kisi.Meslek=veri;
            break;
        }
        case "DogumTarihi":
        {
            kisi.DogumTarihi=Convert.ToDateTime(veri);
            break;
        }
    }
}

private void txtAd_TextChanged(object sender, System.EventArgs e)
{
    Degistir("Ad",txtAd.Text);
}

private void txtSoyad_TextChanged(object sender, System.EventArgs e)
{

```

```

        Degistir("Soyad",txtSoyad.Text);
    }
    private void txtDogumTarihi_TextChanged(object sender, System.EventArgs e)
    {
        Degistir("DogumTarihi",txtDogumTarihi.Text.ToString());
    }
    private void txtMeslek_TextChanged(object sender, System.EventArgs e)
    {
        Degistir("Meslek",txtMeslek.Text);
    }
    private void btnGuncelle_Click(object sender, System.EventArgs e)
    {
        int id;
        id=Convert.ToInt32(lblKisiID.Text.ToString());
        Guncelle(id);
    }

```

Görüldüğü gibi kodlarımız gayet basit. Şimdi güncelleme işlemlerimizi gerçekleştireceğimiz kodları yazalım. Kullanıcımız, TextBox kontrollerinde yaptığı değişikliklerin veritabanınada yansıtılmasını istiyorsa Guncelle başlıklı button kontrolüne tıklayacaktır. İşte kodlarımız.

```

private void btnGuncelle_Click(object sender, System.EventArgs e)
{
    /* Güncelleme işlemi, şu anda ekranda olan Kişi için yapılacağından, bu kişiye ait KisiID sini ilgili Label kontrolümüzden alıyoruz
ve Guncelle isimli metodumuza parametre olarak gönderiyoruz. Asıl güncelleme işlemi Guncelle isimli metodumuzda yapılıyor. */
    int id;
    id=Convert.ToInt32(lblKisiID.Text.ToString());
    Guncelle(id);
}

public void Guncelle(int ID)
{
    /* Sql Server'ımıza bağlantımızı oluşturuyoruz.*/
    SqlConnection conFriends=new SqlConnection("data source=localhost;integrated security=sspi;initial catalog=Friends");
    /* Update sorgumuzu oluşturuyoruz. Dikkat edecek olursanız alanlara atanacak değerler, kisi isimli nesnemizin özelliklerinin
değerleridir. Bu özellik değerleri ise, TextBox kontrollerinin TextChanged olaylarına eklediğimiz kodlar ile sürekli güncel
tutulmaktadır. En ufak bir değişiklik dahi buraya yansıyabilecektir.*/
    string sorgu="Update Kisiler Set Ad='"+kisi.Ad+"',Soyad='"+kisi.Soyad+"',Meslek='"+kisi.Meslek+"',DogumTarihi='"+
kisi.DogumTarihi.ToShortDateString()+"' Where KisiID="+ID;
    SqlCommand cmd=new SqlCommand(sorgu,conFriends); /* SqlCommand nesnemizi sql cümleciğimiz ve geçerli bağlantımız ile
oluşturuyoruz. */

```



```

conFriends.Open(); /* Bağlantımızı açıyoruz.*/
try
{
    cmd.ExecuteNonQuery(); /* Komutumuzu çalıştırıyoruz.*/
}
catch
{
    MessageBox.Show("Başarısız");
}
finally /* Update işlemi herhangi bir neden ile başarısız olsada, olmasada sonuç olarak(finally) açık olan SqlConnection
bağlantılarımızı kapatıyoruz. */
{
    conFriends.Close();
}
}

```

İşte uygulama kodlarımız bu kadar. Şimdi gelin uygulamamızı çalıştırıp deneyelim. Öncelikle KisiID değeri 1000 olan satıra ait verileri getirelim.

Şekil 3. KisiID=1000 Kaydına ait veriler Kisi nesnemize yüklenir.

Şimdi verilerde bir kaç değişiklik yapalım ve güncelleyelim. Ben Ad alanında yer alan "S." değerini "Selim" olarak değiştirdim. Bu durum sonucunda yapılan değişikliklerin veritabanına yazılıp yazılmadığını ister programımızdan tekrar 1000 nolu satırı getirerek bakabiliriz istersekde Sql Server'dan direkt olarak bakabiliriz. İşte sonuçlar.

KisiID	Ad	Soyad	DogumTarihi	Meslek
1000	Burak Selim	Senyurt	12.04.1976	Matematik Müh.
*				

Şekil 4. Güncelleme işleminin sonucu.

Programımız elbette gelişmeye çok, ama çok açık. Örneğin kodumuzda hata denetimi yapmadığımız bir çok ölü nokta var. Bunların geliştirilmesini siz değerli okurlarımıza bırakıyorum. Bu makalemizde özetle, bir arayüzü bir sınıfa nasıl uyguladığımızı, bu arayüzü nasıl yazdığımızı hatırlamaya çalıştık. Ayrıca, sınıfımıza ait bir nesne örneğine, bir tablodaki belli bir veri satırına ait verileri nasıl alabileceğimizi, bu nesne özelliklerinde yaptığımız değişiklikleri tekrar nasıl veri tablosuna gönderebileceğimizi inceledik. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Checked, Unchecked Anahtar Kelimeleri ve Overflow Hatası

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, değişkenlerin içerdikleri verilerin birbirleri arasında atanması sırasında oluşabilecek durumları incelemeye çalışacağız. Bildiğiniz gibi, değişkenler bellekte tutulurken, tanımlandıkları veri tipine göre belirli bir bit boyutuna sahip olurlar. Ayrıca her değişkenimizin belli bir değer aralığı vardır. Programlarımızı yazarken, çoğu zaman değişkenleri birbirlerine atarız. Küçük boyutlu bir değişkeni, kendisinden daha büyük boyutlu bir değişkene atarken bir problem yoktur. Ancak, boyutu büyük olan bir değişkeni, daha küçük boyuta sahip bir değişkene atamak istediğimizde durum değişir. Elbette böyle bir durumda, derleyicimiz bizi uyaracaktır. Ancak bilinçli olarak yani tür dönüştürme anahtar kelimelerini kullandığımız durumlarda herhangi bir derleyici hatasını almayız. Bu konuyu daha iyi anlayabilmek, değişkenleri tanımladığımız türlere ait boyut bilgilerinin iyi bilinmesini gerektirir. Bu amaçla aşağıdaki tabloda, C# programlama dilinde kullanılan değişken türlerini bulabilirsiniz.

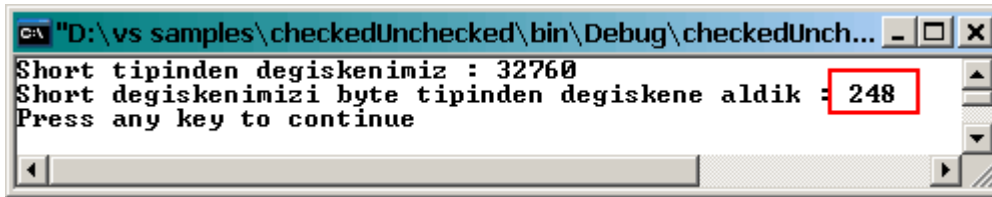
Değişken Türü	Boyut (Bit)	Alt Aralık	Üst Aralık
Byte	8	0	255
SByte	8	-128	127
Short	16	-32768	32767
UShort	16	0	65535
Int	32	-2,147,483,648	2,147,483,647
UInt	32	0	4,294,967,295
Long	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
ULong	64	0	18,446,744,073,709,551,615
Float	32	+/- 1.5 X 10 ⁻⁴⁵	+/- 3.4 X 10 ³⁸
Double	64	+/- 5 X 10 ⁻³²⁴	+/- 1.7 X 10 ³⁰⁸
Decimal	128	1 X 10 ⁻²⁸	7.9 X 10 ¹⁰²⁸
Char	16	-	-
Bool	-	-	-

Tablo 1. C# Değişken Türlerini Hatırlayalım.

Büyük alanlı değişkenlerin, küçük alanlı değişkenler içine alınması sırasında neler olabileceğini gözlemlemek amacıyla aşağıdaki örneğimizi inceleyelim.

```
using System;
namespace checkedUnchecked
{
    class Class1
    {
        static void Main(string[] args)
        {
            short degisken1=32760;
            byte degisken2;
            degisken2=(byte)degisken1;
            Console.WriteLine("Short tipinden değişkenimiz : {0}",degisken1);
            Console.WriteLine("Short değişkenimizi byte tipinden degiskene aldık : {0}",degisken2);
        }
    }
}
```

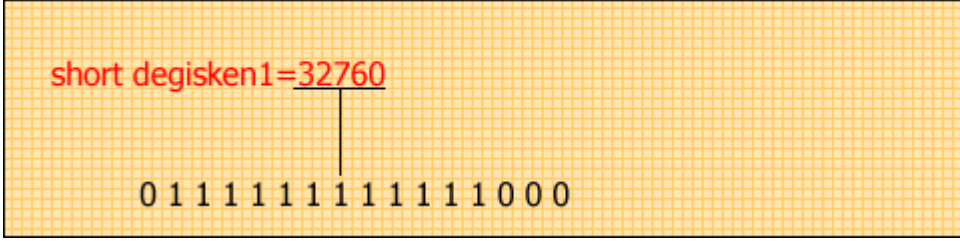
Uygulamamızda, Short tipinde degisken1 isminde bir değişkenimiz var. Değeri 32760. Short tipi değişken türleri -32768 ile 32767 arasındaki değerleri alabilen sayısal bir türdür. Degisken2 isimli , değişkenimiz ise Byte türünden olmakla birlikte değer aralığı 0 ile 255 arasındadır. Kodumuzda bilinçli bir şekilde, (byte) dönüştürücüsü yardımıyla, short türünden değişkenimizi, byte türünden değişkenimize atıyoruz. Bu kod hatasız olarak derlenecektir. Ancak, uygulamamızı çalıştırdığımızda karşımıza çıkacak olan sonuç beklemediğimiz bir sonuç olacaktır.



Şekil 2. Sonuç şaşırtıcı mı?

Gördüğümüz gibi anlamsız bir sonuç elde ettik. Şimdi gelin bunun nedenini ele alalım. Öncelikle degisken1 isimli short türünden değişkenimizi ele alalım. Short tipi 16 bitlik bir veri alanını temsil eder. Degisken1 isimli veri tipimizin bellekte bitset düzeyde tutuluş şekli aşağıdaki gibi olacaktır.

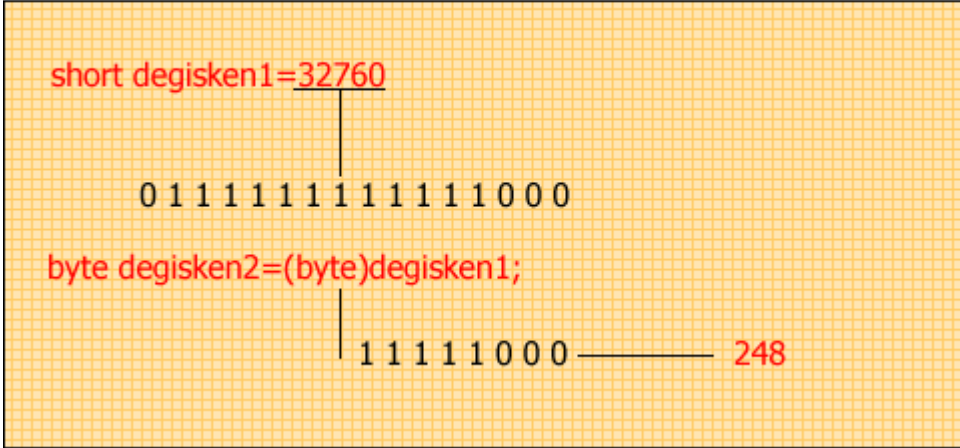
Bellek



Şekil 2. Short tipinden değişkenimizin bellekte tutuluşu.

Şimdi byte türünden tanımladığımız değişkenimizi ele alalım. Byte türü 8 bitlik bir veri türüdür ve 0 ile 256 arasında sayısal değerler alır. degisken1 isimli short türünden değişkenimizi, byte türünden degisken2 değişkenimiz içine almaya çalıştığımızda aşağıdaki sonuç ile karşılaşırız.

Bellek



Şekil 3. Atama sonrası.

Görüldüğü gibi 16 bitlik short tipi değişkenin ilk 8 biti havaya uçmuştur. Çünkü, byte veri tipi 8 bitlik bir veri tipidir. Dolayısıyla 16 bitlik bir alanı, 8 bitlik alana sığdırmaya çalıştığımızda veri kaybı meydana gelmiş ve istemediğimiz bir sonuç ortaya çıkmıştır. Elbette hiçbirimiz, yazdığımız programların çalışması sırasında böylesi mantık hatalarının olmasını istemeyiz. Bu durumun en güzel çözümlerinden birisi, **checked** anahtar kelimesini kullanmaktır. Checked anahtar kelimesi, uygulandığı bloktaki tüm tür dönüşümlerini kontrol eder ve yukarıdaki gibi bir durum oluştuğunda, Overflow istisnasının fırlatılmasını sağlar. Yukarıdaki örneğimizi şimdide checked bloğu ile çalıştıralım.

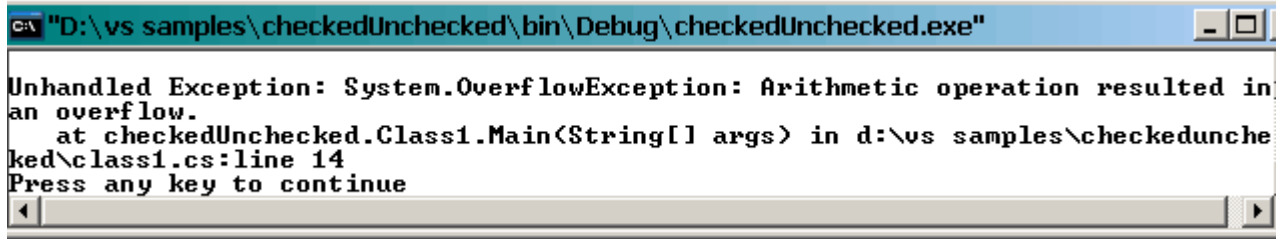
```
static void Main(string[] args)
{
    short degisken1=32760;
    byte degisken2;
    checked
    {
        degisken2=(byte)degisken1;
    }
}
```

```

    }
    Console.WriteLine("Short tipinden değişkenimiz : {0}",degisken1);
    Console.WriteLine("Short değişkenimizi byte tipinden değişkene aldık : {0}",degisken2);
}

```

Bu durumda uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 4. OverFlow istisnasının fırlatılması.

Derleyicimiz, checked anahtar kelimesinin kullanıldığı bloktaki tüm tür dönüşümlerini izlemeye alır. Eğer büyük alanlı bir değişken türü, kendisinden daha küçük alanlı bir değişken türüne atanmaya çalışırsa derleyici, OverFlow istisnasını fırlatır. Bu bize, checked bloklarının, try...catch...finally blokları ile kullanarak, kodumuzu dahada güvenli bir hale getirmemize imkan sağlar. Ne dediğimizi daha iyi anlayabilmek için, yukanda yazdığımız kodu aşağıdaki gibi değiştirelim.

```

static void Main(string[] args)
{
    short degisken1=32760;
    byte degisken2=0;
    try
    {
        checked
        {
            degisken2=(byte)degisken1;
            Console.WriteLine("Short tipinden değişkenimiz : {0}",degisken1);
            Console.WriteLine("Short değişkenimizi byte tipinden değişkene aldık : {0}",degisken2);
        }
    }
    catch(System.OverflowException hata)
    {
        Console.WriteLine(hata.Message.ToString());
        Console.WriteLine("Değişken 2 :{0}",degisken2.ToString());
    }
}

```

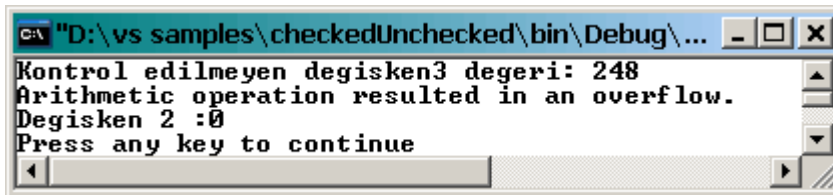
Burada, checked bloğunu, try...catch...finally bloğu içine alarak, programın kesilemesinin de önüne geçmiş olduk. Bununla birlikte, checked anahtar kelimesinin bir diğer özelliğide, kontrol altına aldığı blok içerisinde oluşabilecek taşma hatalarının sonucunda,

taşma hatasına maruz kalan değişkenlerin orjinal değerlerini korumasıdır. Örneğin yukarıdaki örnekte, byte türündeki değişkenimiz 248 değeri yerine ilk atama yaptığımız 0 değerini korumuştur.

Diğer yandan bazen, meydana gelebilecek bu tarz taşma hatalarını görmezden gelerek, bazı tür atamalarının mutlaka yapılmasını isteyebiliriz. Bu durumda unchecked bloklarını kullanırız. Bunu daha iyi anlayabilmek için, aşağıdaki örneğimize bir göz atalım.

```
static void Main(string[] args)
{
    short degisken1=32760;
    byte degisken2=0;
    byte degisken3=0;
    try
    {
        checked
        {
            unchecked
            {
                degisken3=(byte)degisken1;
                Console.WriteLine("Kontrol edilmeyen degisken3 değeri: {0}",degisken3);
            }
            degisken2=(byte)degisken1;
            Console.WriteLine("Short tipinden değişkenimiz : {0}",degisken1);
            Console.WriteLine("Short değişkenimizi byte tipinden değişkene aldık : {0}",degisken2);
        }
    }
    catch(System.OverflowException hata)
    {
        Console.WriteLine(hata.Message.ToString());
        Console.WriteLine("Değişken 2 :{0}",degisken2.ToString());
    }
}
```

Uygulamamızı çalıştırdığımızda, degisken3 isimli byte türünden değişkenimiz için, bilinçli olarak gerçekleştirdiğimiz dönüşümün gerçekleştiğini görebiliriz. Bunu sağlayan unchecked anahtar kelimesidir. Dolayısıyla oluşacak OverFlow hatasının görmezden gelindiğini görürüz.



Şekil 5. Unchecked anahtar kelimesinin uygulanmasının sonucu.

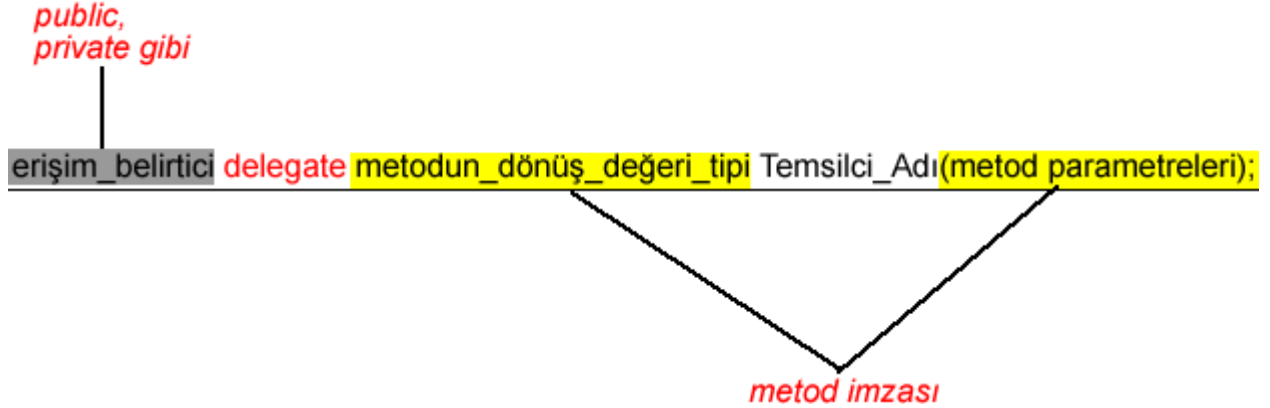
Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Temsilciler (Delegates) Kavramına Giriş

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, C# programlama dilinde ileri seviye kavramlardan biri olan Temsilcileri(delegates) incelemeye başlayacağız. Temsilciler ileri seviye bir kavram olmasına rağmen, her seviyeden C# programcısının bilmesi gereken unsurlardandır. Uygulamalarımızı temsilciler olmadan da geliştirebiliriz. Ancak bu durumda, yapamıyacıklarımız, yapabileceklerimizin önüne geçecektir. Diğer yandan temsilcilerin kullanımını gördükçe bize getireceği avantajları daha iyi anlayacağımız kanısındayım. Bu makalemizde temsilcileri en basit haliyle anlamaya çalışacağız.

Temsilci (delegate), program içerisinde bir veya daha fazla metodu gösteren(işaret eden), referans türünden bir nesnedir. Programlarımızda temsilciler kullanmak istediğimizde, öncelikle bu temsilcinin tanımını yaparız. Temsilci tanımları, arayüzlerdeki metod tanımlamaları ile neredeyse aynıdır. Tek fark delegate anahtar sözcüğünün yer almasıdır. Bununla birlikte, bir temsilci tanımlandığında, aslında işaret edebileceği metod(ların) imzalarından belirlemiş olur. Dolayısıyla, bir temsilciyi sadece tanımladığı metod imzasına uygun metodlar için kullanabileceğimizi söyleyebiliriz. Temsilci tanımları tasarım zamanında yapılır. Bir temsilciyi, bir metodu işaret etmesi için kullanmak istediğimizde ise, çalışma zamanında onu new yapılandırıcısı ile oluşturur ve işaret etmesini istediğimiz metodu ona parametre olarak veririz. Bir temsilci tanımı genel haliyle, aşağıdaki şekilde gibidir.



Şekil 1. Temsilci tanımlaması.

Şekildende görüldüğü gibi, temsilciler aslında bir metod tanımlarlar fakat bunu uygulamazlar. İşte bu özellikleri ile arayüzlerdeki metod tanımlamalarına benzerler. Uygulamalarımızda, temsilci nesneleri ile göstermek yani işaret etmek istediğimiz metodlar bu imzaya sahip olmalıdır. Bildiğiniz gibi metod imzaları, metodun geri dönüş tipi ve aldığı parametreler ile belirlenmektedir.

Bir temsilcinin tanımlanması, onu kullanmak için yeterli değildir elbette. Herşeyden önce bir amacımız olmalıdır. Bir temsilciyi çalışma zamanında oluşturabiliriz ve kullanabiliriz. Bir temsilci sadece bir tek metodu işaret edebileceği gibi, birden fazla metod için tanımlanmış ve oluşturulmuş temsilcileride kullanabiliriz. Diğer yandan, tek bir temsilcide birden fazla temsilciyi toplayarak bu temsilcilerin işaret ettiği, tüm metodları tek bir seferde çalıştırma lüksüne de sahibizdir. Ancak temsilciler gerçek anlamda iki amaçla kullanılırlar. Bunlardan birincisi olaylardır(events). Diğer yandan, bugünkü makalemizde işleyeceğimiz gibi, bir metodun çalışma

zamanında, hangi metodların çalıştırılacağına karar vermesi gerektiği durumlarda kullanırız. Elbette bahsetmiş olduğumuz bu amacı, herhangi bir temsilye ihtiyaç duymadan da gerçekleştirebiliriz. Ancak temsilcileri kullanmadığımızda, bize sağladığı üstün programlama tekniği, kullanım kolaylığı ve artan verimliliği göz ardı etmiş oluruz.

Şimdi dilerseniz bahsetmiş olduğumuz bu amaçla ilgili bir örnek verelim ve konuyu daha iyi kavramaya çalışalım. Örneğin, personelimizin yapmış olduğu satış tutarlarına göre, prim hesabı yapan ve ilgili yerlere bu değişiklikleri yazan bir projemiz olsun. Burada primlerin hesaplanması için değişik katsayılar, yapılan satışın tutarına göre belirlenmiş olabilir. Örneğin bu oranlar düşük, orta ve yüksek olarak tanımlanmış olsun. Personel hangi gruba giriyorsa, metodumuz ona uygun metodu çağırırsın. İşte bu durumda karar verici metodumuz, çalıştırabileceği metodları temsil eden temsilci nesnelerini parametre olarak alır. Yani, çalışma zamanında ilgili metodlar için temsilci nesneleri oluşturulur ve karar verici metoda , hangi metod çalıştırılacak ise onun temsilcisi gönderilir. Böylece uygulamamız çalıştığında, tek yapmamız gereken hangi metodun çalıştırılması isteniyorsa, bu metoda ilişkin temsilcinin, karar verici metoda gönderilmesi olacaktır.

Oldukça karışık görünüyor. Ancak örnekleri yazdıkça daha iyi kavrayacağınıza inanıyorum. Şimdiki örneğimizde, temsilcilerin tasarım zamanında nasıl tanımlandığını, çalışma zamanında nasıl oluşturulduklarını ve karar verici bir metod için temsilcilerin nasıl kullanılacağını incelemeye çalışacağız.

using System;

namespace Delegates1

{

public class Calistir

{

public static int a;

public delegate void temsilci(int deger); /* Temsilci tanımlamamızı yapıyoruz. Aynı zamanda temsilcimiz , değer döndürmeyen ve integer tipte tek bir parametre alan bir metod tanımlıyor. Temsilcimizin adı ise temsilci.*/

/* Şimdi bu temsilciyi kullanacak bir metod yazıyoruz. İşte karar verici metodumuz budur. Dikkat ederseniz metodumuz parametre olarak, temsilci nesnemiz tipinden bir temsilci(Delegate) alıyor. Daha sonra metod bloğu içinde, parametre olarak geçirilen bu temsilcinin işaret ettiği metod çağırılıyor ve bu metoda parametre olarak integer tipte bir değer geçiriliyor. Kısaca, metod içinden, temsilcinin işaret ettiği metod çağırılıyor. Burada, temsilci tanımına uygun olan metodun çağırılması garanti altına alınmıştır. Yani, programın çalışması sırasında, new yapılandırıcısı kullanarak oluşturacağımız bir temsilci(delegate), kendi metod tanımı ile uyuşmayan bir metod için yaratılmaya çalışıldığında bir derleyici hatası alacağız. Dolayısıyla bu, temsilcilerin yüksek güvenlikli işaretçiler olmasını sağlar. Bu , temsilcileri, C++ dilindeki benzeri olan işaretçilerden ayıran en önemli özelliktir. */

public void Metod1(Calistir.temsilci t)

{

t(a);

}

}

class Class1

{

/* İkiKat ve UcKat isimli metodlarımız, temsilcimizin programın çalışması sırasında işaret etmesini istediğimiz metodlar. Bu nedenle imzaları, temsilci tanımımızdaki metod imzası ile aynıdır. */

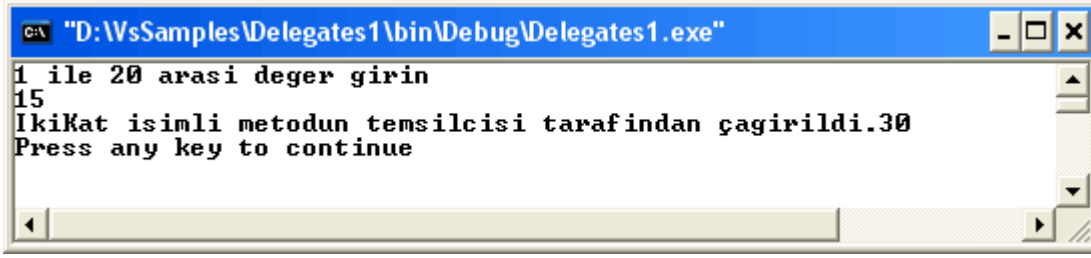

```

        public static void IkiKat(int sayi)
        {
            sayi=sayi*2;
            Console.WriteLine("IkiKat isimli metodun temsilcisi tarafından çağırıldı."+sayi.ToString());
        }
        public static void UcKat(int sayi)
        {
            sayi=sayi*3;
            Console.WriteLine("UcKat isimli metodun temsilcisi tarafından çağırıldı."+sayi.ToString());
        }
        static void Main(string[] args)
        {
            /* Temsilci nesnelerimiz ilgili metodlar için oluşturuluyor. Burada, new yapılandırıcısı ile oluşturulan temsilci nesneleri
            parametre olarak, işaret edecekleri metodun ismini alıyorlar. Bu noktadan itibaren t1 isimli delegate nesnemiz IkiKat isimli metodu,
            t2 isimli delegate nesnemizde UcKat isimli metodu işaret ediceklerdir. */
            Calistir.temcilci t1=new Delegates1.Calistir.temcilci(IkiKat);
            Calistir.temcilci t2=new Delegates1.Calistir.temcilci(UcKat);
            Console.WriteLine("1 ile 20 arası değer girin");
            Calistir.a=System.Convert.ToInt32(Console.ReadLine());
            Calistir c=new Calistir();

            /* Kullanıcının Console penceresinden girdiği değer göre, Calistir sınıfının a isimli integer tipteki değerini 10 ile
            karşılaştırılıyor. 10 dan büyükse, karar verici metodumuza t1 temsilcisi gönderiliyor. Bu durumda Metod1 isimli karar verici
            metodumuz, kendi kod bloğu içinde t1 delegate nesnesinin temsil ettiği IkiKat metodunu, Calistir.a değişkeni ile çağırıyor. Aynı
            işlem tarzı t2 delegate nesnesi içinde geçerli.*/
            if(Calistir.a>=10)
            {
                c.Metod1(t1);
            }
            else
            {
                c.Metod1(t2);
            }
        }
    }
}

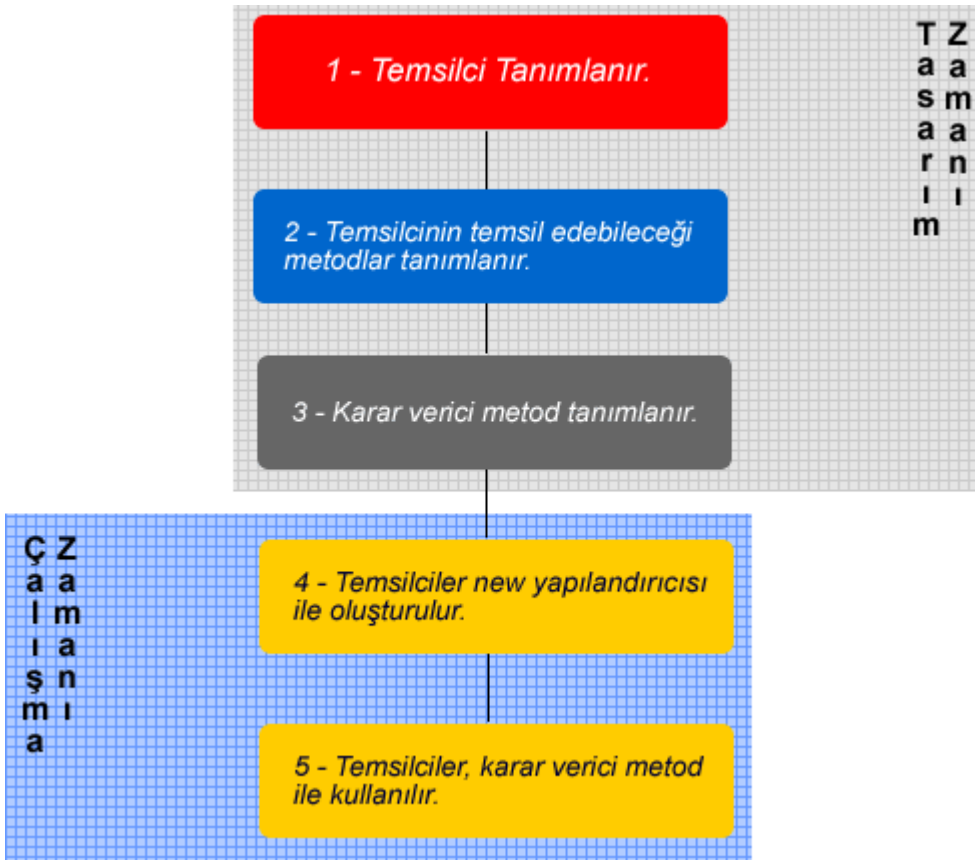
```

Uygulamamızı çalıştıralım ve bir değer girelim.



Şekil 2. Programın çalışmasının sonucu.

Bu basit örnek ile umarım temsilciler hakkında biraz olsun bilgi sahibi olmuşsunuzdur. Şimdi temsilciler ile ilgili kavramlarımıza devam edelim. Yukarıdaki örneğimiz ışığında temsilcileri programlarımızda temel olarak nasıl kullandığımızı aşağıdaki şekil ile daha kolay anlayabileceğimizi sanıyorum.



Şekil 3. Temsilcilerin Karar Verici metodlar ile kullanımı.

Yukarıdaki örneğimizde, her bir metod için tek bir temsilci tanımladık ve temsilcileri teker çağırdık. Bu Single-Cast olarak adlandırılmaktadır. Ancak programlarımız da bazen, tek bir temsilciye birden fazla temsilci ekleyerek, birden fazla metodu tek bir temsilci ile çalıştırmak isteyebiliriz. Bu durumda Multi-Cast temsilciler tanımlarız. Şimdi multi-cast temsilciler ile ilgili bir örnek yapalım. Bu örneğimizde t1 isimli temsilcimiz, multi-cast temsilcimiz olacak.

```
using System;
```

```

namespace Delegates2
{
    public class temsilciler
    {
        public delegate void dgTemsilci(); /* Temsilcimiz tanımlanıyor. Geri dönüş değeri olmayan ve parametre almayan metodları temsil edebilir. */
        /* Metod1, Metod2 ve Metod3 temsilcilerimizin işaret etmesini istediğimiz metodlar olacaktır.*/
        public static void Metod1()
        {
            Console.WriteLine("Metod 1 çalıştırıldı.");
        }
        public static void Metod2()
        {
            Console.WriteLine("PI değeri 3.14 alınsın");
        }
        public static void Metod3()
        {
            Console.WriteLine("Mail gönderildi...");
        }
        /* Temsilcilerimizi çalıştıran metodumuz. Parametre olarak gönderilen temsilciyi, dolayısıyla bu temsilcinin işaret ettiği metodu alıyor. */
        public static void TemsilciCalistir(temsilciler.dgTemsilci dt)
        {
            dt(); /* Temsilcinin işaret ettiği metod çalıştırılıyor.*/
        }
    }
    class Class1
    {
        static void Main(string[] args)
        {
            /* Üç metodumuz içinde temsilci nesnelerimiz oluşturuluyor.*/
            temsilciler.dgTemsilci t1=new Delegates2.temsilciler.dgTemsilci(temsilciler.Metod1);
            temsilciler.dgTemsilci t2=new Delegates2.temsilciler.dgTemsilci(temsilciler.Metod2);
            temsilciler.dgTemsilci t3=new Delegates2.temsilciler.dgTemsilci(temsilciler.Metod3);
            Console.WriteLine("sadece t1");
            temsilciler.TemsilciCalistir(t1);
            Console.WriteLine("---");
            /* Burada t1 temsilcimize, t2 temsilcisi ekleniyor. Bu durumda, t1 temsilcimiz hem kendi metodunu hemde, t2 temsilcisinin işaret ettiği metodu işaret etmeye başlıyor. Bu halde iken TemsilciCalistir metodumuza t1 temsilcisini göndermemiz her

```

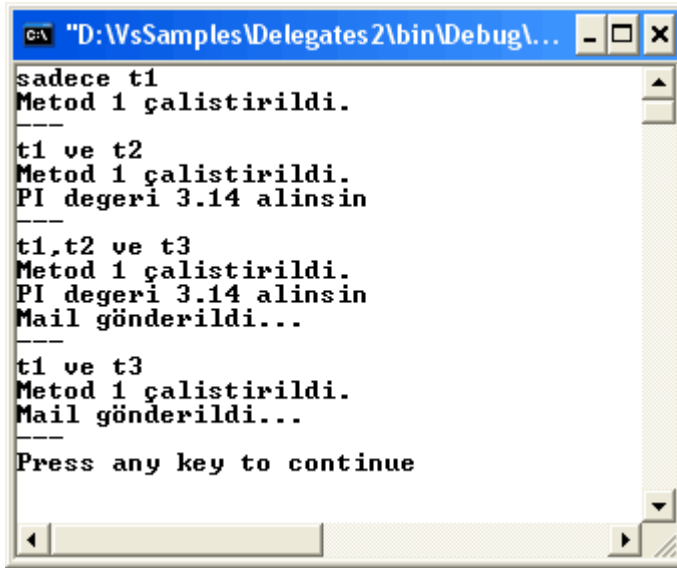
iki temsilcinin işaret ettiği metodların çalıştırılmasına neden oluyor.*/

```

        t1+=t2;
        Console.WriteLine("t1 ve t2");
        temsilciler.TemsilciCalistir(t1);
        Console.WriteLine("---");
        t1+=t3; /* Şimdi t1 temsilcimiz hem t1, hem t2, hem de t3 temsilcilerinin işaret ettiği metodları işaret etmiş olacak.*/
        Console.WriteLine("t1,t2 ve t3");
        temsilciler.TemsilciCalistir(t1);
        Console.WriteLine("---");
        t1-=t2; /* Burada ise t2 metodunu t1 temsilcimizden çıkartıyoruz. Böylece, t1 temsilcimiz sadece t1 ve t3 temsilcilerini
        içeriyor. */
        Console.WriteLine("t1 ve t3");
        temsilciler.TemsilciCalistir(t1);
        Console.WriteLine("---");
    }
}

```

Uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 4. Multi-Cast temsilciler.

Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde temsilcilerin kullanıldığı olaylar(events) kavramına gireceğiz. Hepinize mutlu günler dilerim.

Net Data Providers(Veri Sağlayıcıları)

Değerli Okurlarım, Merhabalar.

Bugünkü makalemiz ile, ADO.NET ' te yer alan veri sağlayıcılarını inceleyeceğiz. Bildiğiniz gibi hepimiz uygulamalarımızda yoğun bir şekilde veri kaynaklarını kullanmaktayız. Normalde sistemimizde, bu veri kaynaklarına erişmek için kullanılan sistem sürücüleridir. Bu sürücüler, sistemimize dll kütüphaneleri olarak yüklenirler ve kendilerini sisteme kayıt ederler(register). Bu noktadan itibaren bu veri sürücülerinin içerdiği fonksiyonları kullanarak veritabanları üzerinde istediğimiz işlemleri gerçekleştirebiliriz. Kısaca, bu veri sürücülerini uygulamalarımız ile, veritabanı arasındaki iletişimi sağlarlar. Sistemizide yüklü olan programlara göre pek çok veri sürücüsüne sahip olabiliriz. Örneğin ODBC sürücüler, SQL sürücüler, Ole Db Jet sürücüler ve bazıları.

ADO.NET ile veritabanı uygulamaları geliştirirken, bu sürücüler üzerinden veritabanlarına erişim sağlarız. Bu sebeple .Net Framework 'te her bir veri sürücüsü için geliştirilmiş veri sağlayıcıları (data providers) vardır. Bu veri sağlayıcılarının görevi, uygulamalarımız ile veri sürücülerini bağlamak ve veri sürücülerindeki ilgili kütüphane fonksiyonlarını çalıştırarak veriler üzerinde işlem yapabilmemizi sağlamaktır. .Net Framework'ün 1.1 sürümü aşağıdaki listede yer alan veri sağlayıcıları ile birlikte gelmektedir. .Net Framework'ün ilk sürümlerinde sadece Sql ve Ole Db veri sağlayıcıları varsayılan olarak yer almaktadır. Ancak 1.1 sürümü ile birlikte bu veri sağlayıcılarına, Oracle ve ODBC veri sağlayıcıları da eklenmiştir.

.Net Framework Veri Sağlayıcıları
Data Provider For SQL Server
Data Provider For OLE DB
Data Provider For ODBC
Data Provider For Oracle

Tablo 1: .NET Veri Sağlayıcıları

Şimdi dilerseniz, bu veri sağlayıcıları kısaca incelemeye çalışalım.

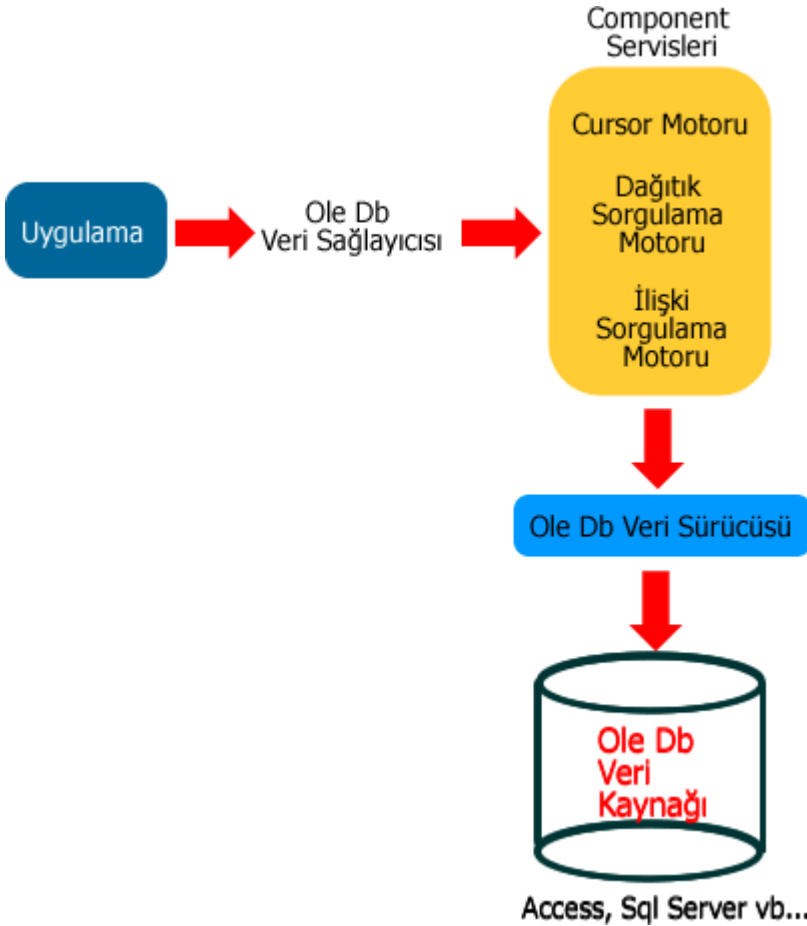
SQL veri sağlayıcısına ait tüm üyeler, System.Data.SqlClient isim uzayında yer almaktadır. SQL veri sağlayıcısının en önemli özelliği, sql motoruna direkt sql api'si üzerinden erişim sağlayabilmesidir. Bu özellik ona diğer veri sağlayıcılarına göre daha yüksek performans kazandırır. Nitekim sql veri sağlayıcısı, sql server'a doğrudan ulaşmak için kendi iletişim protokolü olan TDS(Tabular Data Stream)'yi kullanmaktadır. Elbette bu özelliği ile, örneğin SqlDataReader nesnesinin kullanıldığı veri okuma yöntemlerinde, ole db veri kaynağına göre çok daha hızlı ve verimlidir. Nitekim aynı sql veri kaynaklarına ole db veri sağlayıcısı ile de erişmemiz mümkündür. Ama belirttiğimiz gibi performans ve verimlilik bu iki veri kaynağı için oldukça farklıdır.



Şekil 1. Sql Veri Sağlayıcımız.

Sql veri sağlayıcısı, Sql Server'ın 7 ve daha üstü versiyonlarını desteklemektedir. Bu nedenle 6.5 versiyonu ve daha öncesi için, Ole Db veri sağlayıcısını kullanmak zorundayız. Diğer yandan Sql veri sağlayıcısı MDAC(Microsoft Data Access Component)'ın 2.6 veya üstü sürümünün sistemimizde kurulu olmasını gerektirmektedir. Sql veri sağlayıcısı, sql server'ın 7.0 ve sonraki sürümlerinde özellikle çok katlı uygulamalarda yüksek verim ve performans sağlar.

Ole Db veri sağlayıcısı, Ole Db desteği veren tüm veri sürücülerini ilişki kurabilmektedir. Bunu yaparken, Ole Db Com nesnelerini kullanır. Aşağıdaki şekilde görüldüğü gibi, uygulamamızda ole db veri sağlayıcısı kullanarak, bir oledb veri kaynağına erişmek oldukça maliyetlidir. Bunun yanında ole db'yi destekleyen çok çeşitli veri kaynağı sürücülerinin olması ole db nin ürün yelpazesini genişliğini gösterir.



Şekil 2 . Ole Db Veri Sağlayıcısı

Ole Db veri sağlayıcısı Ole Db desteği veren her türlü veri sürücüsü ile çalışabilir. Aşağıda ole db veri sağlayıcısı ile kullanılabilen örnek Ole Db veri sürücülerini listelenmiştir.

Microsoft Jet 4.0 OLE DB Provider
 Microsoft OLE DB Provider For Data Mining Services
 Microsoft OLE DB Provider for DTS Packages
 Microsoft OLE DB Provider for Indexing Service
 Microsoft OLE DB Provider for Internet Publishing
 Microsoft OLE DB Provider for ODBC Drivers
 Microsoft OLE DB Provider for Olap Services 8.0
 Microsoft OLE DB Provider for Oracle
 Microsoft OLE DB Provider for Outlook Search
 Microsoft OLE DB Provider for SQL Server
 Microsoft OLE DB Simple Provider

Ole Db veri sağlayıcısının .net framework üyeleri, System.Data.OleDb isim uzayında yer alır. Çoğunlukla bu veri sağlayıcısını Access tablolarına erişmek için uygulamalarımızda kullanmaktayız. Bununla birlikte Paradox, dBASE, Excel, FoxPro, Oracle 7.3, Oracle8 gibi veri tablolarına da erişebiliriz. Diğer yandan Oracle sürücülerine ve ODBC sürücülerine de erişebiliriz. Ancak elbetteki, çok katlı uygulamalarda, sql veri sağlayıcısını veya oracle veri sağlayıcısını tercih etmemiz daha doğru olacaktır. Diğer yandan ole db veri sağlayıcısı, com servsileri ile veri sürücülerine eriştiği için, özellikle sql veri sağlayıcısına göre çok daha düşük bir performans sergiler. Ole Db veri kaynakları ile çalışan ole db veri sağlayıcısının , özellikle sql server'ın 6.5 ve önceki sürümlerinin kullanıldığı tek katlı ve çok katlı uygulamalarda kullanılması tercih edilir. Bununla birlikte, access tabloları ile çalışırken, çok katlı mimarilerin, bu veri tabloları üzerinden ole db sağlayıcıları ile oluşturulması microsoft otoriterlerince tavsiye edilmemektedir.

ODBC veri sağlayıcısı, Ole Db veri sağlayıcısı gibi, ODBC desteği veren sürücüler ile, ODBC Servis Component'lerini kullanarak iletişim kurar.



Şekil 3 . ODBC Veri Sağlayıcısı

ODBC veri sağlayıcısı ile ilgili üyeler, .net framework içinde, System.Data.Odbc isim uzayında yer almaktadır. Aslında bu veri sağlayıcı, .net framework'ün 1.0 versiyonunda yer almamaktaydı. Ancak 1.1 versiyonu ile birlikte ADO.NET ' teki yerini almıştır. ODBC sürücüsü yardımıyla, sql server'a, access tablolarına ve odbc'yi destekleyen veri sürücülerine erişebiliriz. ODBC veri sağlayıcısı, odbc veri kaynakları üzerinden yapılan tek katlı (single-tier) ve orta katlı(middle-tier) mimarilerinde kullanılabilir.

Oracle servis sağlayıcısı, .net framework'ün System.Data.OracleClient isim uzayında yer alan üyelerden oluşur. Oracle servis sağlayıcısı, oracle veri kaynaklarına erişebilmek için, sql veri sağlayıcısı gibi kendi iletişim protokolünü içeren Oracle Client Connectivity'yi kullanır. Oracle veri sağlayıcısının .net'e yerleştirilmesindeki temel amaç, oracle veri tabanlarına ole db veri sağlayıcısı ile ole db üzerinden değil, doğrudan erişilebilmesini sağlamaktır. Bu sayede oracle veri kaynağı ile oluşturulan etkileşimde en iyi performansın elde edilmesi sağlanmıştır. Zaten bu yönü ile oracle veri sağlayıcısı, sql veri sağlayıcısına benzer bir yapıdadır. Doğal olarak, oracle veri kaynakları üzerinde gerçekleştirilen, çok katlı ve tek katlı mimarilerde yüksek performans sergilemektedir.

Bu makalemizde .net veri sağlayıcılarına kısaca değinmeye çalıştık. İlerleyen makalelerimiz ile birlikte ado.net'in tüm kavramlarını incelemeye çalışacağız. Bir sonraki makalemde ole db veri sağlayıcısı üyelerinden olan, OleDbConnection nesnesini incelemeye çalışacağız. Hepinize mutlu günler ve iyi çalışmalar dilerim.

Sql Tablolarına Resim Ekleme

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, örnek bir sql tablomuzda yer alan image veri tipinden bir alana, seçtiğimiz resim dosyalarını nasıl kaydedebileceğimizi incelemeye çalışacağız. Öncelikle, sql tablolarında kullanabildiğimiz image tipinden biraz bahsedelim. Bu tip, ikili (binary) formatta verileri tutmak için geliştirilmiştir. Image veri tipi, 0 byte' dan 2,147,483,647 byte'a kadar veri alanını taşıyabilmektedir. Bu alan, verileri bir byte dizisi şeklinde tutmaktadır. Dolayısıyla resim dosyalarını tutmak için ideal bir yapı sergiler. Elbette üst sınırı aşmamaya çalışmak gerekir. Çoğu zaman uygulamalarımızda, resim dosyalarını ikili bir dizi şeklinde sql tablolarımızda, image tipindeki alanlarda tutmak isteyeceğimiz durumlar oluşabilir. (Örneğin şirket çalışanları ile ilgili personel bilgilerini tuttuğumuz tablolarda, personelin vesikalık fotoğraflarını bu alanlarda taşıdığımızı düşünelim.)

İşte şimdi, bu tarz resim dosyalarını, sql tablolarımızdaki ilgili alanlara nasıl yazabileceğimizi inceleyeceğiz. Yapmamız gereken işlem aslında son derece kolay. Resim dosyasını ikili formatta okumak, dosyanın okunan byte'larını bir byte dizisine aktarmak ve oluşan bu diziye, image tipindeki alanımıza aktarmak. Bu anafikir ışığında işlemlerimizi gerçekleştirebilmek için, öncelikle dosyamızı bir FileStream nesnesine açacağız. Daha sonra, bir BinaryReader nesnesi kullanarak, FileStream nesnesinin işaret ettiği dosyadan tüm byte'ları okuyacak ve bunları bir byte dizisine aktaracağız. Sonrada oluşturduğumuz bu diziye, sql tablomuzda yer alan image veri tipindeki alana koyacağız.

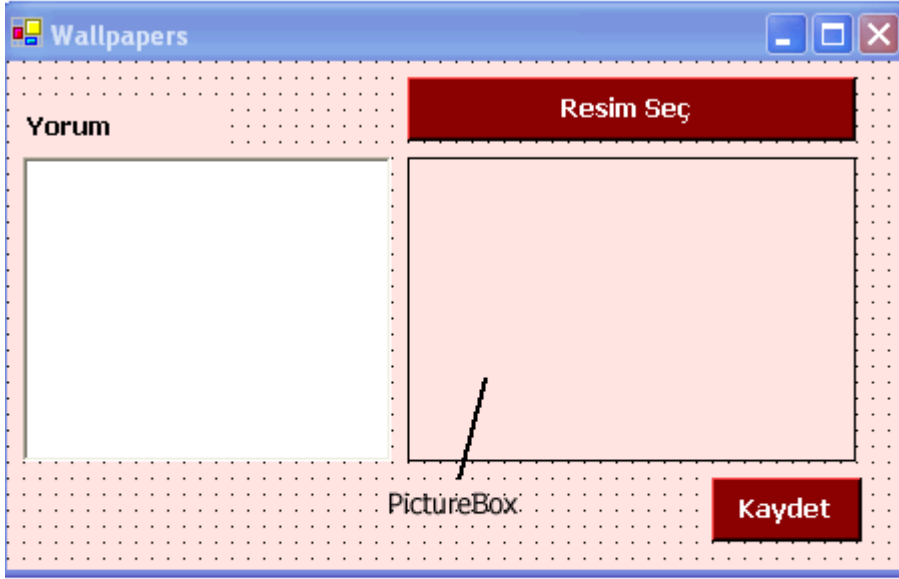
Uygulamamızı gerçekleştirmeden önce, FileStream ve BinaryReader sınıfları hakkında da kısaca bilgi verelim. FileStream nesnelerini, sistemimizde yer alan dosyaları okumak veya bu dosyalara yazmak amacıyla kullanırız. BinaryReader nesnesi, FileStream nesnesinden byte türünden bir akış oluşturmamızı sağlar. BinaryReader, FileStream nesnesinin temsil ettiği dosyadan, okumanın yönlendirileceği kaynağa doğru bir akım oluşturur. Bu akış sayesinde, FileStream nesnesinin temsil ettiği dosyadan verileri byte byte okuyabilir ve bir byte dizisine aktarabiliriz. Bunun nasıl yapıldığını örneğimizi geliştirdiğimizde daha iyi anlayacağız.

Şimdi dilerseniz, uygulamamızı geliştirmeye başlayalım. Öncelikle, veri tablomuzu yapalım. Örneğin, internetten indirip bilgisayarımızda bir klasörde topladığımız güzel duvar kağıtlarını tablomuzda kaydetmek istediğimizi varsayalım. Bununla ilgili olarak aşağıdaki örnek tabloyu oluşturdum.

Key	ID	Name	Data Type	Size(...)	Nulls	Default
		WallID	int	4	☐	
		Yorum	char	250	☐	
		Resim	image	16	☐	

Şekil 1. Wallpapers tablomuz

Sıra geldi uygulamamızın ekranını tasarlamaya. Uygulamamızda kullanıcı, istediği resim dosyasını seçecek, bunu aynı zamanda ekranda yer alan bir PictureBox kontrolünde görebilecek ve bunu isterse Wallpapers isimli sql tablomuza yazabilecek. İşte ekran tasarımı.



Şekil 2. Form Tasarımımız.

Şimdi uygulama kodlarımızı yazmaya başlayabiliriz.

```
string resimAdresi; /* OpenFileDialog kontrolünden seçtiğimiz dosyanın tam adresini tutacak genel bir degisken. */
```

```
/* Bu metodumuzda OpenFileDialog kontrolümüzün temel ayarlarını yapıyoruz. */
```

```
public void DialogHazirla()
```

```
{
```

```
    ofdResim.Title="Duvar Kagidini Seç"; /* Dosya açma iletişim kutumuzun başlığını belirliyoruz. */
```

```
    ofdResim.Filter="Jpeg Dosyaları(*.jpg)|*.jpg|Gif dosyaları(*.gif)|*.gif"; /* İletişim kutumuzun, sadece jpg ve gif dosyalarını göstermesini, Filter özelliği ile ayarlıyoruz.*/
```

```
}
```

```
private void Form1_Load(object sender, System.EventArgs e)
```

```
{
```

```
    DialogHazirla();
```

```
}
```

```
private void btnResimSec_Click(object sender, System.EventArgs e)
```

```
{
```

```
    /* Kullanıcı bu butona tıkladığında, OpenFileDialog kontrolümüz, dosya açma iletişim kutusunu açar. Kullanıcı bir dosya seçip OK tusunda bastığında, Picture Box kontrolümüze seçilen resim dosyası alınarak gösterilmesi sağlanır. Daha sonra seçilen dosyanın tam adresi label kontrolümüze alınır ve resimAdresi degiskenimize atanır. */
```

```
    if(ofdResim.ShowDialog()==DialogResult.OK)
```

```

{
    pbResim.Image=System.Drawing.Image.FromFile(ofdResim.FileName); /* Drawing isim uzayında yer alan Image sınıfının
    FromFile metodunu kullanarak belirtilen adresteki dosya PictureBox kontrolü içine çizilir. */
    lblDosyaAdi.Text=ofdResim.FileName.ToString();
    resimAdresi=ofdResim.FileName.ToString();
}
}
private void btnKaydet_Click(object sender, System.EventArgs e)
{
    /* Simdi en önemli kodlarımızı yazmaya baslıyoruz. Öncelikle dosyamızı açmamız gerekli. Çünkü resim dosyasının içeriğini byte
    olarak okumak istiyoruz. Bu amaçla FileStream nesnemizi oluşturuyor ve gerekli parametrelerini ayarlıyoruz. İlk parametre,
    dosyanın tam yolunu belirtir. İkinci parametre ise dosyamızı açmak için kullanacağımızı belirtir. Son parametre ise dosyanın okuma
    amacı ile açıldığını belirtir. */
    FileStream fsResim=new FileStream(resimAdresi,FileMode.Open,FileAccess.Read);
    /* BinaryReader nesnemiz, byte dizimiz ile, parametre olarak aldığı FileStream nesnesi arasında , veri akisini sağlamak için
    oluşturuluyor. Akim, FileStream nesnesinin belirttiği dosyadan, dosyadaki byte'ların aktarılacağı diziye doğru olacaktır.*/
    BinaryReader brResim=new BinaryReader(fsResim);
    /* Simdi resim adında bir byte dizisi oluşturuyoruz. brResim isimli BinaryReader nesnemizin, ReadBytes metodunu kullanarak,
    bu nesnenin veri akisi için bağlantı kurduğu FileStream nesnesinin belirttiği dosyadaki tüm byte'lari, byte dizimize aktarıyoruz.
    Böylece resim dosyamızın tüm byte'lari yani dosyamızın kendisi, byte dizimize aktarılmış oluyor.*/
    byte[] resim=brResim.ReadBytes((int)fsResim.Length);
    /* Son olarak, BinaryReader ve FileStream nesnelerini kapatıyoruz. */
    brResim.Close();
    fsResim.Close();
    /* Artık Sql bağlantımızı oluşturabilir ve sql komutumuzu çalıştırabiliriz. Önce SqlConnection nesnemizi oluşturuyoruz. */
    SqlConnection conResim=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=sspi");
    /* Simdi sql komutumuzu çalıştıracak olan SqlCommand nesnemizi oluşturuyoruz. Burada alanların değerlerini parametreler
    üzerinden aktardığımıza dikkat edelim. */
    SqlCommand cmdResimKaydet=new SqlCommand("insert into Wallpapers (Yorum,Resim) values (@yorum,@res)",conResim);
    cmdResimKaydet.Parameters.Add("@Yorum",SqlDbType.Char,250).Value=txtYorum.Text; /* Bu parametremiz @Yorum
    isminde ve Char tipinde. 250 karakter uzunluğunda. Hemen aynı satırda Value özelliğini kullanarak parametrenin değerinde
    belirtiyoruz. */
    cmdResimKaydet.Parameters.Add("@res",SqlDbType.Image,resim.Length).Value=resim; /* Seçtiğimiz resim dosyasının
    byte'larını, tablodaki ilgili alana taşıyacak parametremizi belirtiyoruz. Değer olarak, resim isimli byte dizimizi aktarıyoruz. Parametre
    tipinin, image olduğuna dikkat edelim. */
    /* Günlük blogumuzda, Sql bağlantımızı açıyoruz. Ardından, sql komutumuzu ExecuteNonQuery metodu ile çalıştırıyoruz. Son
    olarakta herhangi bir hata olsada, olmasada, finally blogunda sql bağlantımızı kapatıyoruz.*/
    try
    {

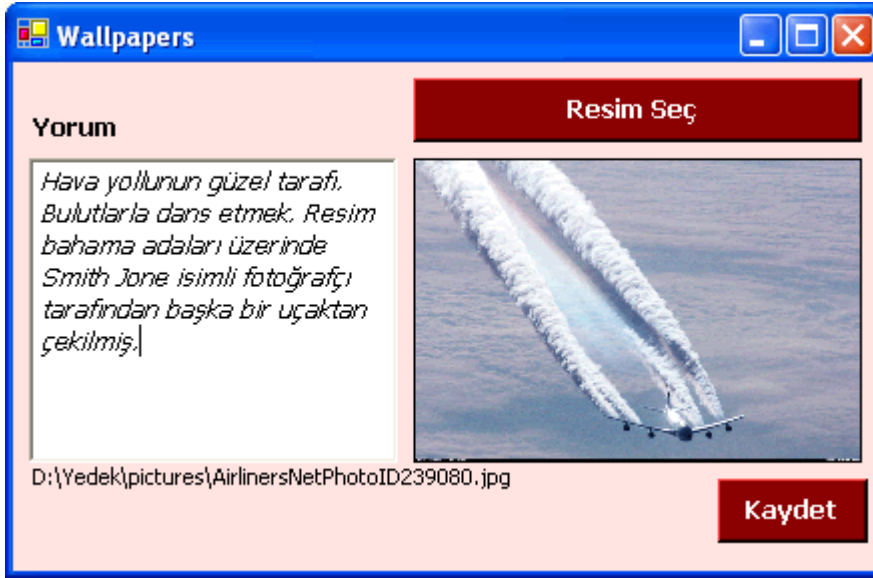
```

```

conResim.Open();
cmdResimKaydet.ExecuteNonQuery();
MessageBox.Show(lblDosyaAdi.Text+" Tabloya Basarili Bir Sekilde Kaydedildi.");
}
catch(Exception ex)
{
    MessageBox.Show(ex.Message.ToString());
}
finally
{
    conResim.Close();
}
}

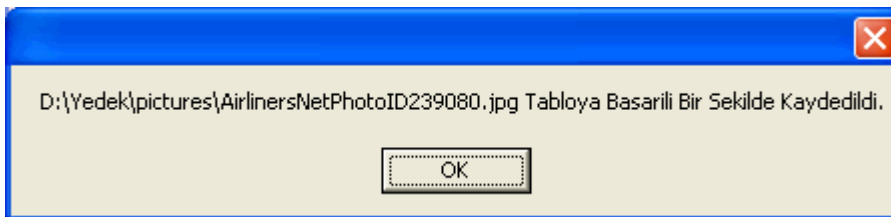
```

Şimdi uygulamamızı çalıştıralım ve tablomuzdaki Resim alanına yazmak için bir resim seçelim.



Şekil 3. Resim Seçilir.

Şimdi Kaydet butonuna tıklayalım. Bu durumda aşağıdaki mesajı alırsınız.



Şekil 4. Resim image tipindeki alana kaydedildi.

Son olarak sql tablomuzda bu alanların nasıl görüldüğüne bir bakalım.

Yorum	Resim
1001	Hava yollunun güzel tarafı. Bulutlarla dans etmek. <Binary>

Şekil 5. Tablonun görünümü.

Gördüğünüz gibi, tablomuzda Resim adlı image tipindeki alanımızda <Binary> yazmaktadır. Acaba gerçekten resmimiz bu alana düzgün bir şekilde kaydedildi mi? Bir sonraki makalemizde, bu kez var olan image alanlarını, tablodan nasıl okuyacağımızı ve bir dosyaya nasıl kaydedebileceğimizi incelemeye çalışacağız. Umuyorumki hepiniz için yararlı bir makale olmuştur. Bir sonraki makalemizde görüşmek dileğiyle mutlu günler, iyi çalışmalar dilerim.

Sql Tablolarındaki Binary Resimlere Bakmak ve Dosya Olarak Kaydetmek

Değerli Okurlarım, Merhabalar.

Hatırlayacağınız gibi bir önceki makalemizde, bir resim dosyasını sql tablosundaki Image veri tipinden bir alana nasıl yazabileceğimizi görmüştük. Bugünkü makalemizde ise, bu tablodaki Image veri tipindeki Resim alanında yer alan byte'lara nasıl okuyabileceğimizi ,(Örneğimizde, PictureBox kontrolünde nasıl görüntüleyebileceğimizi inceledik) ve bu alandaki resmi, jpg uzantılı bir resim dosyası olarak nasıl kaydedebileceğimizi incelemeye çalışacağız.

Image tipindeki binary(ikili) bir alandan verileri okumak için yine stream nesnelerinden ve BinaryWriter sınıfından faydalanacağız. Visual Studio.Net ortamında, SqlClient isim uzayındaki sınıfları kullanarak Wallpapers isimli sql tablomuza eriştiğimizde, PictureBox kontrolüne , Image tipindeki Resim alanımızı bağlayamadığımızı görürüz. Bu nedenle, bu ikili(binary) alanı okuyup, PictureBox kontrolümüzün anlayacağı bir hale getirmeliyiz.

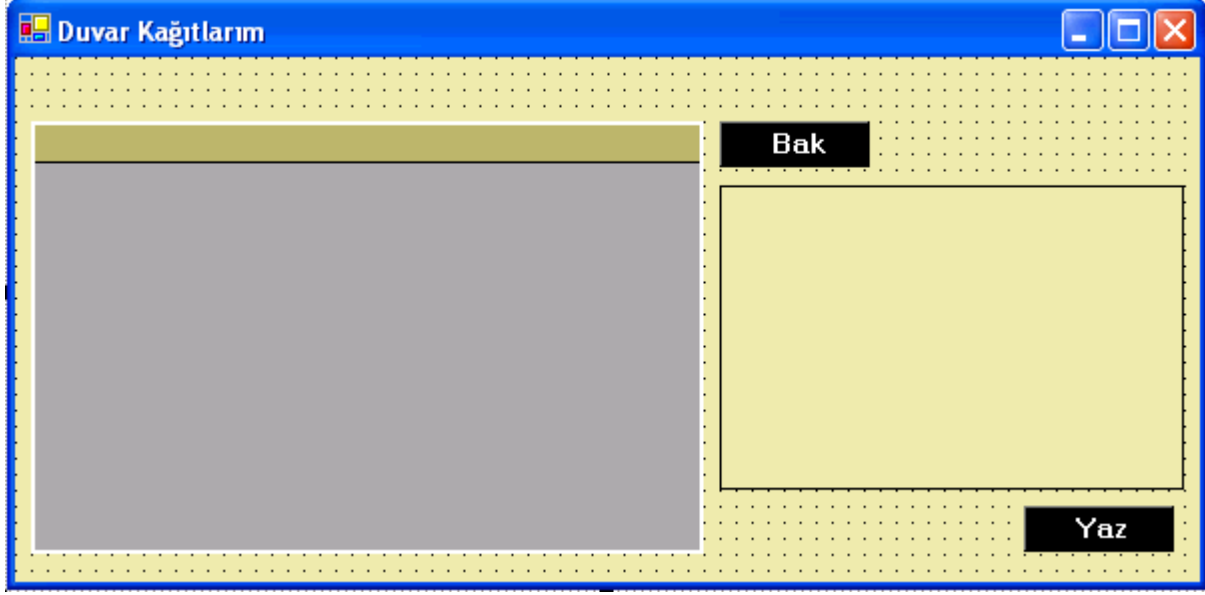
Bu amaçla, bu iki alandaki veriyi okuyucak ve bunu bellekteki bir tampon bölgeye alacağız . Daha sonra bellekte oluşturduğumuz bu MemoryStream alanını, System.Drawing.Image sınıfının FromStream metoduna parametre olarak vereceğiz. Böylece, PictureBox kontrolümüzün Image özelliği, resmin içeriğini bellekteki bu tampon alandan okuyabilecek. Dolayısıyla resmimiz gösterilebilecek.

Ancak burada dikkat etmemiz gereken başka bir husus var. O da bu ikili alanı nasıl okuyacağımız. Bu alanı ikili olarak okuyabilmek için, SqlDataReader nesnesine SequentialAccess parametresini vereceğiz. Bu parametre SqlDataReader nesnesinin, verileri sırasal bir şekilde okuyabilmesine imkan sağlamaktadır. Normalde SqlDataReader okuduğu veri satırını komple alır ve alanların indeksleri sayesinde ilgili verilere ulaşılır. Bununla birlikte SqlDataReader nesnemizin sadece ileri yönlü ve yalnız okunabilir bir veri akışı sağladığını hatırladığınızı sanıyorum. Bu sırasal okuma yeteneği sayesinde, makalemize konu olan tablonun, Resim adındaki ikili alanının tüm byte'larını sırasal bir şekilde okuyabilme imkanına sahip olacağız.

Kullanacağımız teknik ise biraz uzun bir kodlama gerektirmekle birlikte, pek çok konuyada açıklık getirmektedir. Yapacağımız işlem şudur. Sql tablomuzdan kullanıcının seçtiği satıra ait Resim alanını bir SqlDataReader nesnesi ile elde etmek. Daha sonra, bu alanda sırasal bir okuma başlatıp, tüm byte'ları, BinaryWriter nesnesi yardımıyla bloklar halinde, bir MemoryStream nesnesine aktarmak. Son olarak PictureBox kontrolümüze, bellekteki tampon bölgede tutulan bu akımı aktararak resmin görüntülenebilmesini sağlamak. MemoryStream nesneleri bellekte geçici olarak oluşturulan byte dizilerine işaret eder. Doğrudan bellekte oluşturuldukları için performans açısından yüksek verimlilik sağlarlar. Çoğunlukla programlarımızda oluşturduğumuz geçici dosyalar için MemoryStream oldukça etkin bir yöntemdir. Diğer bir deyişle programlarımızda geçici depolamalar yapmak için idealdir.

Aynı teknik yardımıyla, kullanıcı seçtiği resmi bir dosya olarak kaydedebilecek. Bu kez, MemoryStream nesnesi yerine, fiziki bir dosyayı temsil edecek FileStream nesnesini kullanacağız. Bu konular biraz karışık gibi görünsede, kodun içindeki detaylı açıklamalar

sayesinde olayı iyice kafanızda canlandırabileceğinize inanıyorum. Şimdi derseniz uygulamamızın ekranını tasarlayalım ve ardından kodlarımızı yazalım.



Şekil 1. Form Tasarımımız.

```
SqlConnection conResim; /* SqlConnection nesnemizi tanımlıyoruz. */
private void Form1_Load(object sender, System.EventArgs e)
{
    conResim=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=sspi"); /* SqlConnection
nesnemizi oluşturuyor ve Northwind veritabanına bağlanıyoruz. */
    SqlDataAdapter daResim=new SqlDataAdapter("Select WallID,Yorum From Wallpapers",conResim); /* Wallpapers tablosundan,
WallID, ve Yorum alanlarının değerlerini almak ve bunları SqlDataAdapter nesnemizin Fill metodu ile DataTable nesnemizin bellekte
işaret ettiği alana aktarmak için SqlDataAdapter nesnemizi oluşturuyoruz. */
    DataTable dtResim=new DataTable("Duvarlar"); /* DataTable nesnemizi oluşturuyoruz.*/
    daResim.Fill(dtResim); /* DataTable'nesnemizi select sorgusu sonucu elde edilen veri satırları ile dolduruyoruz. */
    dgResim.DataSource=dtResim; /* DataGridView nesnemizi DataTable veri kaynağına bağlıyoruz. */
}
/* Yaz başlıklık buton kontrolüne tıklandığında, kullanıcının seçtiği resim sistemimize, jpg uzantılı bir resim dosyası olarak
kaydedilcektir.*/
private void btnYaz_Click(object sender, System.EventArgs e)
{
    /* SqlDataReader nesnemiz, ileri yönlü bir okuma sağlamak için kullanılacak. */
    SqlDataReader drResim;
    int secilen;
    /* Kullanıcının dataGridView kontrolünde seçtiği satırın, ilk sütununu yani WallID değerini seçiyoruz. Bu değeri sql sorgumuzda, ilgili
satıra ait resim alanını bulmak için kullanacağız. */
```

```

secilen=System.Convert.ToInt32(dgResim[dgResim.CurrentRowNumber,0].ToString());
/* Sql Sorgumuzu oluřturuyoruz. Bu sorgu, seilen satıra ait Resim alanının deėerini elde etmemizi saėlıyacak.*/
string sqlStr="Select Resim From Wallpapers Where WallID="+secilen;
SqlCommand cmdResim=new SqlCommand(sqlStr,conResim); /* SqlCommand nesnemizi, sql sorgumuz ve SqlConnection
nesnemiz zerinden alıřtırılmak zere oluřturuyoruz. */
/* Eėer SqlConnection'ımız aık deėilse aıyoruz. Nitekim SqlCommand nesnesinin ierdiėi sql sorgusunun alıřtırılması aık bir
baėlantıyı gerektirmektedir. */
if(conResim.State!=ConnectionState.Open)
{
    conResim.Open();
}
/* SqlDataReader nesnemizi, SqlCommand nesnemizin, ExecuteReader metodu ile dolduruyoruz.
CommandBehavior.SequentialAccess parametresi sayesinde, Resim alanı zerinde byte seviyesinde sırasal bilgi okuma imkanına
sahip oluyoruz. */
drResim=cmdResim.ExecuteReader(CommandBehavior.SequentialAccess);
/* Resim alanındaki byte'ları taşıyacak bir dizi oluřturuyoruz. Bu dizinin boyutu 50. BinaryWrite nesnemiz , FileStream
nesnesinin iřaret ettiėi dosyaya doėru bu dizideki byte'ları akıtacak. Yani seilen Resim alanındaki byte'ları 50 byte'lık bloklar halinde
okuyacaėız ve bu dizileri sırasıyla, BinaryWriter nesnemiz ile, sistemde yazmak zere oluřturduėumuz dosyaya aktaracaėız. Burada
ben 50 byte'lık blokları seėimsel olarak ele aldım. Sizler bu blokları, 100 byte'lık veya 25 byte'lık veya istediėiniz bir miktarda da
kullanabilirsiniz. */
byte[] bytediziResim=new byte[50];
/* FileStream nesnemiz ile, BinaryWriter nesnesinin okuduėu byte'ları yazacak dosyayı oluřturuyoruz. Dosyamız sistemde daha
nceden var olabilir. Bu durumda terkardan aėılıp stne yazılır. Yok ise bu dosya oluřturulur.Diėer yandan FileAccess.Write
parametresi ile dosyayı, yazmak amacıyla atıėımızı belirtiyoruz. Burada deneme olsun diye Deneme.jpg isimli bir dosya oluřturduk.
Ancak dilerseñiz siz, bu dosya adına WallID alanının deėerini de ekleyerek benzersiz dosyalar oluřturabilirsiniz. Veya kullanıcının bir
dosya ismi girmesini isteyebilirsiniz. Bunun geliřtirilemesini siz deėerli okurlarıma bırakıyorum. */
FileStream fs=new FileStream("c:\\Deneme.jpg",FileMode.OpenOrCreate,FileAccess.Write);
/* BinaryWriter nesnemiz, veri akıřını okuduėu alandan, aldıėı fs parametresinin belirttiėi dosyaya doėru bařlatıyor. */
BinaryWriter bw=new BinaryWriter(fs);
long donenBytelar;
long baslangicIndeksi=0;
/* SqlDataReader nesnemizin dndrdėu satırı okumaya bařlıyoruz. Sorgumuzun sadece Resim alanının deėerini
dndrdėuñu hatırlayalım. */
while(drResim.Read())
{
    /* řimdi Resim alanından ilk 50 byte'lık blm okuyoruz. GetBytes metodunun aldıėı ilk parametre, SqlDataReader'ın
dndrdėu veri kmesindeki Resim alanının indeks deėeridir. İkinci parametre bu alanın hangi byte'ından itibaren okunmaya
bařlayacaėıdır. Bařlangı iin 0'ncı byte'tan itibaren okumaya bařlıyoruz. nc parametre okunan byte'ların hangi Byte disizine
yazılacaėını belirtir. Drdnc parametre bu dizi ierisine dizinin hangi indeksinden itibaren yazılmaya bařlıyacaėını ve beřinci

```

parametrede okunan byte'ların, bu dizi içinde kaç byte'lık bir alana yazılacağını belirtiyor. */

```
donenBytelar=drResim.GetBytes(0,0,bytediziResim,0,50);
```

/* GetBytes metodu SqlDataReader nesnesinden okuyup, bytediziResim dizisine aktardığı byte sayısını geri döndürür. Bu dönen değeri donenBytelar isimli Long tipinde değişkenimizde tutuyoruz. Aşağıdaki döngüyle, okunan byte sayısı 50'ye eşit olduğu sürece, Resim alanından 50 byte'lık bloklar okunmaya devam ediyor. Okundukçada, BinaryWriter nesnemiz bu byte'ları FileStream ile açtığımız dosyaya yazıyor. Farz edelimki 386 byte'lık bir alana sahibiz. 350 byte okunduktan sonra, kalan 36 byte'ta son olarak okunur ve bundan sonrada döngünde çıkmış olur.*/

```
while(donenBytelar==50)
```

```
{
```

```
    bw.Write(bytediziResim);
```

```
    bw.Flush();
```

```
    baslangicIndeksi+=50;
```

```
    donenBytelar=drResim.GetBytes(0,baslangicIndeksi,bytediziResim,0,50);
```

```
}
```

```
/* Bahsettiğimiz 36 bytelık kısımda son olarak buradan yazılır. */
```

```
bw.Write(bytediziResim);
```

```
bw.Flush(); /* Flush metodu, BinaryWriter nesnesinin o an sahip olduğu tampon hafızayı temizler. */
```

```
/* BinaryWriter nesnemiz ve FileStream nesnemiz kapatılıyor. */
```

```
bw.Close();
```

```
fs.Close();
```

```
}
```

```
drResim.Close();
```

```
conResim.Close();
```

```
}
```

/* Seçtiğimiz resmi PictureBox kontrolünde göstermek için aşağıdaki tekniği kullanıyoruz. Bu teknikte bellekte geçici bir tampon bölgeyi MemoryStream nesnesi yardımıyla oluşturuyoruz. Bunun dışında yaptığımız işlemlerin tümü, Yaz başlıklı butona uyguladığımız kodlar ile aynı.*/

```
private void btnBak_Click(object sender, System.EventArgs e)
```

```
{
```

```
    SqlDataReader drResim;
```

```
    int secilen;
```

```
    secilen=System.Convert.ToInt32(dgResim[dgResim.CurrentRowNumber,0].ToString());
```

```
    string sqlStr="Select Resim From Wallpapers Where WallID="+secilen;
```

```
    SqlCommand cmdResim=new SqlCommand(sqlStr,conResim);
```

```
    if(conResim.State!=ConnectionState.Open)
```

```
    {
```

```
        conResim.Open();
```

```
    }
```

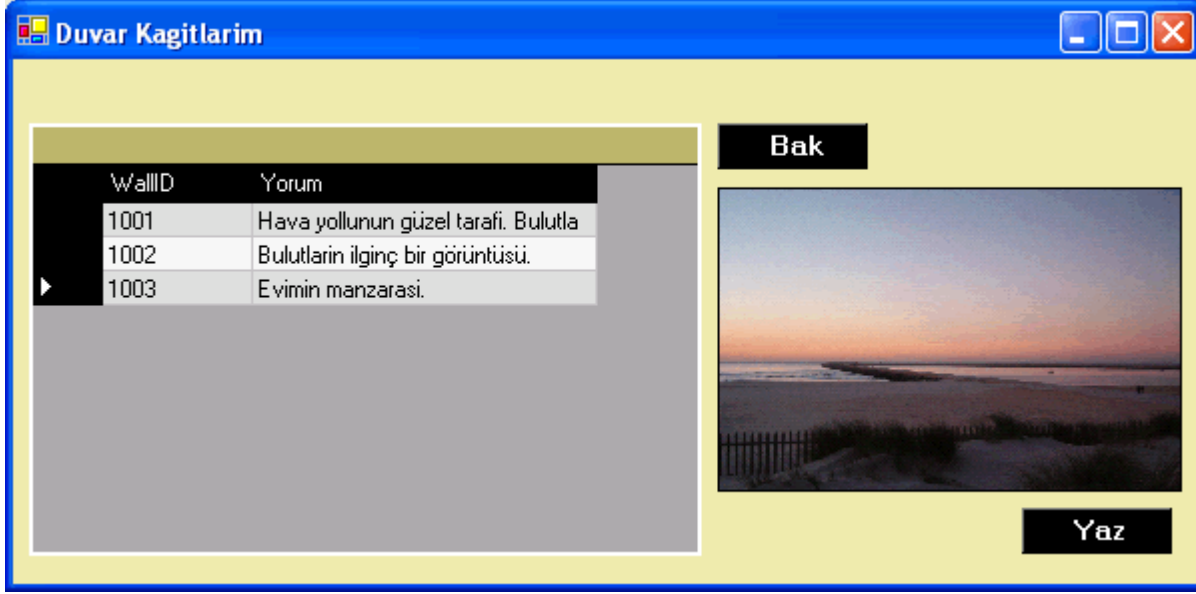
```
    drResim=cmdResim.ExecuteReader(CommandBehavior.SequentialAccess);
```

```

byte[] bytediziResim=new byte[50];
MemoryStream ms=new MemoryStream();
BinaryWriter bw=new BinaryWriter(ms);
long donenBytelar;
long baslangicIndeksi=0;
while(drResim.Read())
{
    donenBytelar=drResim.GetBytes(0,0,bytediziResim,0,50);
    while(donenBytelar==50)
    {
        bw.Write(bytediziResim);
        bw.Flush();
        baslangicIndeksi+=50;
        donenBytelar=drResim.GetBytes(0,baslangicIndeksi,bytediziResim,0,50);
    }
    bw.Write(bytediziResim);
    pbResim.Image=System.Drawing.Image.FromStream(ms); /* Bellekteki tampon bölgeye aldığımız, byte dizisini, PictureBox
kontrolünde göstermek için, Image sınıfının FromStream metodunu kullanıyoruz. Bu metod parametre olarak aldığı akımdan gerekli
byte'ları okuyarak, resmin PictureBox kontrolünde gösterilebilmesini sağlıyor. */
    bw.Flush();
    bw.Close();
    ms.Close();
}
drResim.Close();
conResim.Close();
}

```

Şimdi uygulamamızı çalıştıralım ve herhangi bir resme bakalım. Sonrada bunu kaydedelim.



Şekil 2: WallID değeri 1003 olan satırdaki Resim.

Şimdi Yaz başlıklı butona basıp bu resmi sisteme fiziki bir dosya olarak kaydedelim. Şekildende görüldüğü gibi dosyamız sistemde oluşturulmuştur. Bu dosyaya tıkladığımızda resmimizi görebiliriz.



Şekil 3. Fiziki dosyamız oluşturuldu.

Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

İndeksleyiciler (Indexers)

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde kısaca indeksleyicilerin C# programlama dilindeki kullanımını incelemeye çalışacağız. Bir indeksleyici, bir sınıfı dizi şeklinde kullanabilmek ve bu sınıftan türetilen nesneleri dizinleyebilmek amacıyla kullanılır. Başka bir deyişle bir indeksleyici, nesnelere dizi gibi davranılabilmesini sağlar.

İndeksleyiciler tanımlanışları itibarıyla, özelliklere (properties) çok benzerler . Ancak aralarında temel farklılıklarda vardır. Herşeyden önce bu benzerlik, indeksleyicilerin tanımlanmasında göze çarpar. Bir indeksleyiciyi teorik olarak aşağıdaki söz dizimi ile tanımlanır.

```
public int this[int indis]
{
    get
    {
```

```

    // Kodlar
}
set
{
    // Kodlar
}
}

```

Görüldüğü gibi bir indeksleyici tanımlanması, özellik tanımlanması ile neredeyse aynıdır. Ancak bir indeksleyici tanımlarken uymamız gereken bir takım kurallarda vardır. Bu kurallar aşağıdaki tabloda belirtilmiştir.

İndeksleyici Kuralları
Bir indeksleyici mutlaka bir geri dönüş tipine sahip olmalıdır. Yani bir indeksleyiciyi void olarak tanımlayamayız.
Bir indeksleyiciyi static olarakta tanımlayamayız.
Bir indeksleyici en az bir parametre almalıdır. Bununla birlikte, bir indeksleyici birden fazla ve çeşitte parametrede alabilmektedir.
İndeksleyicileri aşırı yükleyebiliriz (Overload). Ancak bir indeksleyiciyi aşırı yüklediğimizde, bu indeksleyicileri birbirlerinden ayırırken ele aldığımız imzalar sadece parametreler ile belirlenir. İndeksleyicinin geri dönüş değeri bu imzada ele alınmaz.
İndeksleyici parametrelerine , normal değişkenlermiş gibi davranamayız. Bu nedenle bu parametreleri ref ve out anahtar sözcükleri ile yönlendiremeyiz.
Bir indeksleyici her zaman this anahtar sözcüğü ile tanımlamalıyız. Nitekim this anahtar sözcüğü , indeksleyicinin kullanıldığı sınıf nesnelerini temsil etmektedir. Böylece sınıfın kendisi bir dizi olarak kullanılabilir.

Tablo 1. İndeksleyici tanımlama kuralları.

İndeksleyicileri siz değerli okurlarıma anlatmanın en iyi yolunun basit bir örnek geliştirmek olduğunu düşünüyorum. Dilerseniz vakit kaybetmeden örneğimize geçelim.

Öncelikle bir sınıf tanımlayacağız. Bu sınıfımız, Sql veri tabanında oluşturduğumu Personel isimli tablonun satırlarını temsil edebilecek bir yapıda olacak. Tablomuz örnek olarak Personelimize ilişkin ID,Ad,Soyad bilgilerini tutan basit bir veri tablosu. Her bir alan, bahsetmiş olduğumuz sınıf içinde birer özellik olarak tanımlanacak. Diğer yandan başka bir sınıfımızda daha var olacak. Bu sınıfımız ise, bir indeksleyiciye sahip olacak. Bu indeksleyiciyi kullanarak, veri satırlarını temsil eden sınıf örneklerini, bu sınıfı içerisinde tanımlayacağımız object türünden bir dizide tutacağız. Sonuç olarak tablo satırlarına, sınıf dizi elemanlarıymış gibi erişebileceğiz. Burada indeksleyiciler sayesinde sınıfımıza sanki bir diziymiş gibi davranacak ve içerdiği veri satırlarına indeks değerleri ile erişebileceğiz. Örneğimizi geliştirdiğimizde konuyu daha iyi kavrayacağınıza inanıyorum. Şimdi dilerseniz bir console uygulaması açalım ve aşağıdaki kodları yazalım.

```

using System;
using System.Collections;

```

```
using System.Data.SqlClient;
namespace Indexers1
{
    /* Tablomuzda yer alan satırları temsil eden sınıfımızı ve bu tablodaki her bir alanı temsil edecek özelliklerimizi tanımlıyoruz. */
    public class Personel
    {
        private int perid;
        private string perad;
        private string persoyad;
        public int PerID
        {
            get
            {
                return perid;
            }
            set
            {
                perid=value;
            }
        }
        public string PerAd
        {
            get
            {
                return perad;
            }
            set
            {
                perad=value;
            }
        }
        public string PerSoyad
        {
            get
            {
                return persoyad;
            }
            set
            {
```

```

        persoyad=value;
    }
}

```

/* PersonelListesi sınıfımız, Personel tipinden nesneleri tutacak Object türünden bir tanımlar. Bu dizimizde, Personel sınıfı türünden nesneleri tutacağız. Bu nedenle Object türünden tanımladık. Ayrıca sınıfımız bir indeksleyiciye sahip. Bu indeksleyici, object türünden dizimizdeki elemanlara erişirken, bu sınıftan türetilen nesneyi bir diziymiş gibi kullanabilmemize imkan sağlayacak. Yani uygulamamızda, bu sınıftan bir nesne türetilip nesneadi[indis] gibi bir satır yazdığımızda buradaki indis değeri, indeksleyicinin tanımlandığı bloğa aktarılacak ve get veya set blokları için kullanılacak. Bu bloklar aldıkları bu indis parametresinin değerini Object türünden dizimizde kullanarak, karşılık gelen dizi elemanı üzerinde işlem yapılmasına imkan sağlamaktadırlar.*/

```

public class PersonelListesi
{
    private Object[] liste=new Object[10];
    /* Indeksleyicimizi tanımlıyoruz.*/
    public Personel this[int indis]
    {
        get
        {
            /* liste isimli Object türünden dizimizin indis indeksli değerini döndürür. Bunu döndürürken Personel sınıfı tipinden döndürür. Böylece ilgili elemandan Personel sınıfındaki özelliklere, dolayısıyla tablo alanlarındaki değere ulaşmış oluruz. */
            return (Personel)liste[indis];
        }
        set
        {
            /* liste isimli Object türünden dizimizdeki indis indeksli elemana value değerini aktarır. */
            liste[indis]=(Personel)value;
        }
    }
}
class Class1
{
    static void Main(string[] args)
    {
        SqlConnection con=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=sspi");
        SqlCommand cmd=new SqlCommand("Select * From Personel",con);
        SqlDataReader dr;
        con.Open();
        dr=cmd.ExecuteReader();
        PersonelListesi pliste=new PersonelListesi(); /* Indeksleyicimizi kullandığımız sınıftan bir nesne türetiliyoruz. */
    }
}

```

```

int PersonelSayisi=0;
int i=0;
try
{
    /* SqlDataReader nesnesi ile, satırlarımızı okurken bu satıra ait alanların değerlerini tutacak Personel tipinden bir sınıf
    nesnesi oluşturulur ve ilgili alan değerleri bu nesnenin ilgili özelliklerine atanır. */
    while(dr.Read())
    {
        Personel p=new Personel();
        p.PerID=(int)dr[0];
        p.PerAd=dr[1].ToString();
        p.PerSoyad=dr[2].ToString();
        /* Şimdi PersonelListesi sınıfı türünden nesnemize güncel satırı temsil eden Personel nesnesini atıyoruz. Bunu
        yaparken bu sınıf örneğine sanki bir diziymiş gibi davrandığımıza dikkat edelim. İşte bunu sağlayan indeksleyicimizdir. Burada
        PersonelListesi içindeki, object türünden liste isimli dizideki i indeksli elemana p nesnesi aktarılıyor. */
        pliste[i]=p;
        i+=1;
        PersonelSayisi+=1;
    }
    /* Bu döngüde, pliste isimli PersonelListesi türünden nesneye, i indeksini kullanarak içerdiği object türünden liste
    isimli dizi elemanlarına, tanımladığımız indeksleyici sayesinde bir dizi elemanına erişir gibi erişiyoruz. */
    for(int j=0;j<PersonelSayisi;++j)
    {
        /* pliste'nin türetildiği PersonelListesi sınıfı indeksleyicisinin kullandığı dizi elemanlarını Personel türüne
        dönüştürerek elde ettiğimiz için, bu nesnenin özelliklerinede yani veri satırı alanlarınada kolayca erişebiliyoruz. */
        Console.WriteLine(pliste[j].PerAd.ToString()+" "+pliste[j].PerSoyad.ToString()+" "+pliste[j].PerID.ToString());
    }
}
catch(Exception hata)
{
    Console.WriteLine(hata.Message.ToString());
}
finally
{
    dr.Close();
    con.Close();
}
}
}

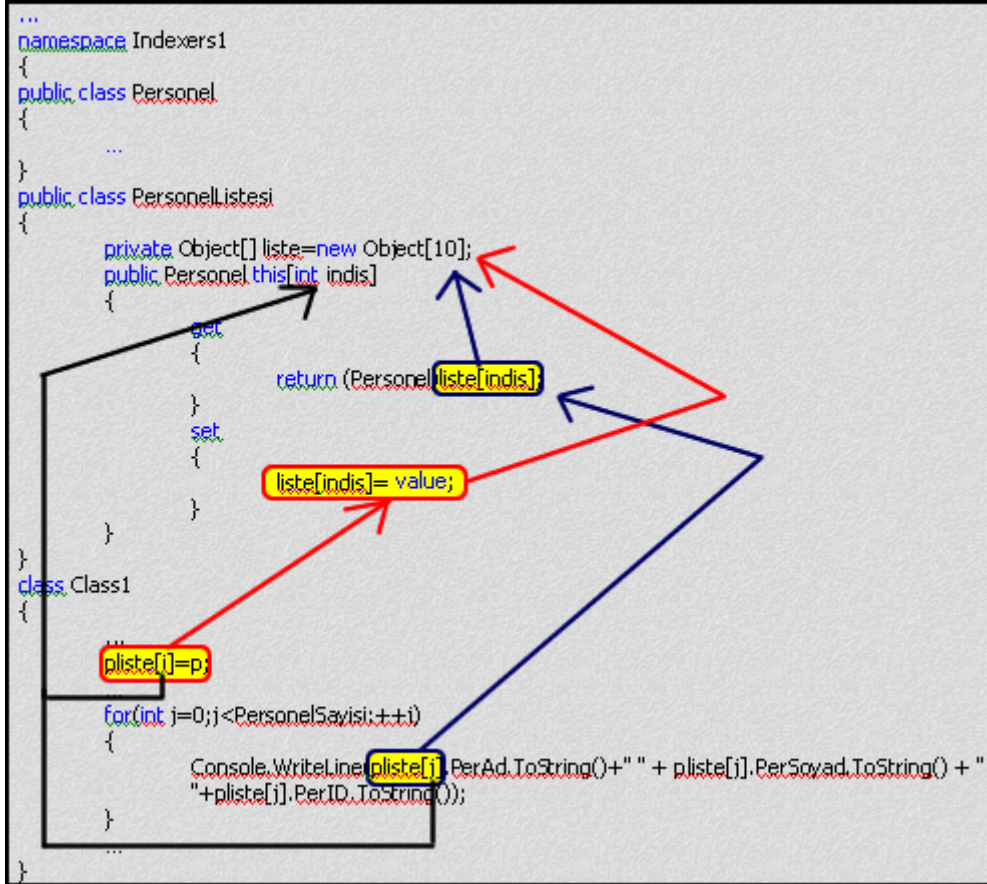
```

```

}

```

Kodlar size karmaşık gelebilir o nedenle aşağıdaki şekil indeksleyicilerin kullanımını daha iyi anlayabilmemizi sağlayacaktır.

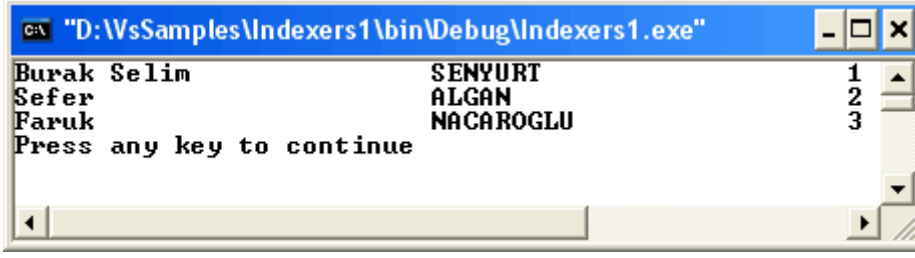


Şekil 1. İndeksleyicilerin Kullanımı

Uygulamamızda `pliste[i]=p;` satırı ile, `Personel` sınıfı türünden nesnemiz, `pliste` isimli `PersonelListesi` sınıfının `i` indeksli elemanı olarak belirleniyor. Bu satır derleyici tarafından işlendiğinde, `PersonelListesi` sınıfındaki indeksleyicimizin `set` bloğu devreye girer. `Set` bloğu `pliste[i]` deki `i` değerini `indis` parametresi olarak alır ve `Object` türünden `liste` isimli dizide `indis` indeksine sahip elemana nesnemizi aktarır.

Diğer yandan, `pliste[j]` ile `PersonelListesi` sınıfına erişildiğinde, indeksleyicinin `get` bloğu devreye girer. `Get` bloğunda, indeksleyicinin parametre olarak aldığı `indis` değeri `j` değeridir. Bu durumda, `liste[indis]` ile, `j` indeksli `liste` dizisi elemanı çağırılır. Sonra bu eleman `Personel` tipine dönüştürülür. Bu sayede `pliste[j].PerAd.ToString()` gibi bir ifade ile, bu nesnenin temsil ettiği özellik değerine ulaşılabilir.

Uygulamamızı çalıştıralım ve deneyelim. Aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 2. Uygulamanın Çalışmasının Sonucu.

Geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Tablo Değişikliklerini GetChanges ile İzlemek

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, bağlantısız olarak veri tabloları ile çalışırken, bu tablolar üzerinde meydana gelen değişiklikleri nasıl izleyebileceğimizi ve davranışlarımızı bu değişikliklere göre nasıl yönlendirebileceğimizi incelemeye çalışacağız.

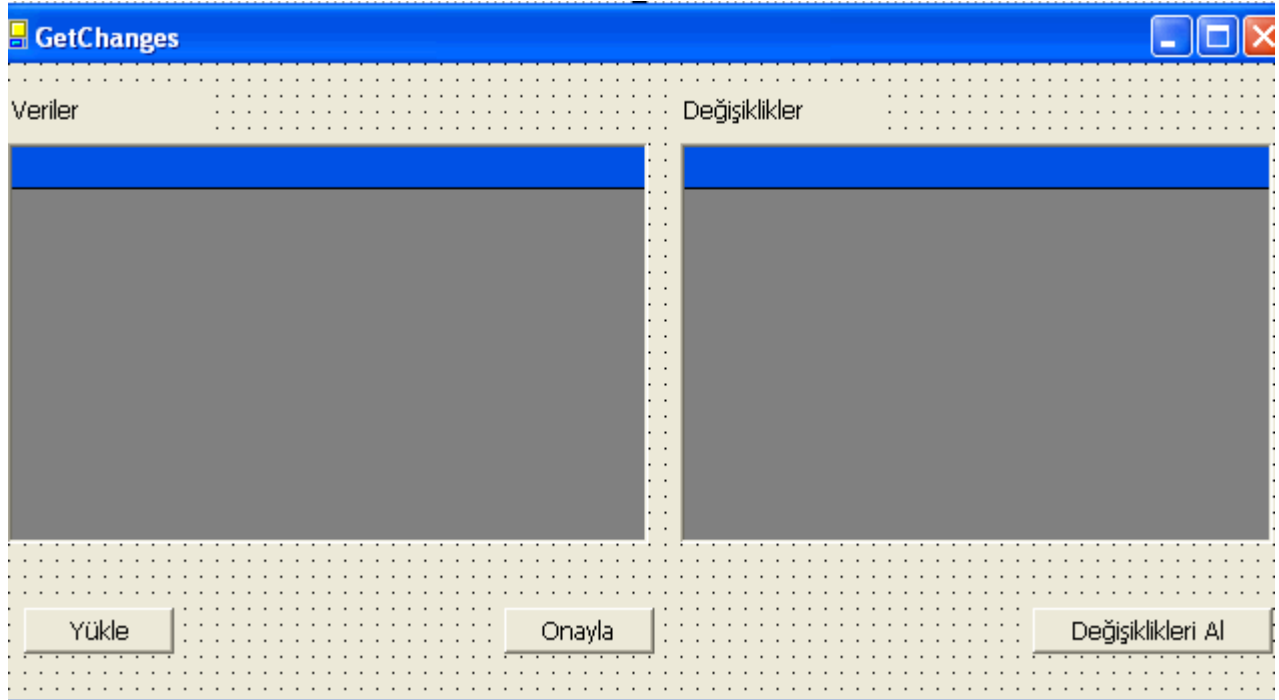
Hepimizin bildiği gibi, bağlantısız veriler ile çalışırken, bir veri kaynağında, makinemizin belleğine tablo veya tabloları alırız. Bu tablolar üzerinde, yeni satırlar oluşturur, var olan satırlar üzerinde değişiklikler yapar, her hangibir satırı siler ve bunlar gibi bir takım işlemler gerçekleştiririz. Tüm bu işlemler, bellek bölgesine aldığımız veriler üzerinde bir DataTable nesnesinde yada bir DataSet kümesinde gerçekleşir. Bununla birlikte, bahsettiğimiz bu değişiklikleri, asıl veri kaynağında yansıtarak, güncellenmelerinide sağlarız.

Ancak, network trafiğinin önemli ve yoğun olduğu uygulamalarda, veri kaynağından aldığımız bir veri kümesinin üzerindeki değişiklikleri, asıl veri kaynağına güncellerken karşımıza iki durum çıkar. İlk olarak, makinemizin belleğinde bulunan tüm veriler asıl veri kaynağına gönderilir ki bu veriler içinde hiç bir değişikliğe uğramamış olanlarda vardır. Diğer yandan istersek, sadece yeni eklenen satırları veya düzenlenen satırları vb., veri kaynağına göndererek daha akılcı bir iş yapmış oluruz. İşte makalemizin ana konusunu teşkil eden bu ikinci durumu gerçekleştirebilmek için GetChanges metodunu kullanırız. GetChanges metodu, DataSet ve DataTable sınıfları içinde kullanılabilir. DataTable ve DataSet sınıfları için, GetChanges metodunun ikişer aşırı yüklenmiş şekli vardır.

DataTable İçin	DataSet İçin
public DataTable GetChanges();	public DataSet GetChanges();
public DataTable GetChanges(DataRowState rowStates);	public DataSet GetChanges(DataRowState rowStates);

Görüldüğü gibi her iki sınıf içinde metodlar aynı şekilde işlemektedir. Sadece metodların geri dönüş tipleri farklıdır. DataSet için, GetChanges metodu başka bir DataSet geri döndürürken, DataTable'ın GetChanges metodu ise geriye DataTable türünden bir nesne referansını döndürür. Peki GetChanges metodunun görevi nedir? GetChanges metodunun parametresiz kullanılan hali, DataTable veya DataSet için, AcceptChanges metodu çağırılana kadar meydana gelen tüm değişiklikleri alır. Örneğin bir DataTable

nesnesinin referans ettiği bellek bölgesinde yer alan bir veri kümesi üzerinde, satır ekleme, satır silme ve satır düzenleme işlemlerini yaptığımızı farzedelim. Bu durumda, bu DataTable nesnesi için AcceptChanges metodunu çağırıp tüm değişiklikleri onaylamadan önce, GetChanges metodunu kullanırsak, tablo üzerindeki tüm değişiklikleri izleyebiliriz. Bunu daha iyi görmek için aşağıdaki örneği inceleyelim. Bu örnekte Sql sunucumuz üzerinde yer alan bir tablo verilerini DataTable nesnemizin bellekte referans ettiği bölgeye yüklüyor ve verilerin görüntüsünü DataGrid kontrolümüze bağlıyoruz. Programdaki önemli nokta, GetChanges metodu ile meydana gelen değişiklikleri başka bir DataTable nesnemize almamızdır. Bu DataTable nesnesinin verileride ikinci DataGrid kontrolümüzde görüntülenecektir. Ancak kullanıcı, DataTable'da meydana gelen bu değişiklikleri DataTable nesnesinin AcceptChanges metodunu kullanarak onayladığında, GetChanges geriye boş bir DataTable nesne referansı döndürecektir. Yani değişiklikleri, AcceptChanges metodu çağırılınca kadar elde edebiliriz. Öncelikle aşağıdaki Formu tasarlayalım.



Şekil 1. Form Tasarımımız.

Şimdide program kodlarımızı oluşturalım.

```
SqlConnection conNorthwind; /*Sql sunucumuza yapacağımız bağlantıyı sağlayacak SqlConnection nesnemizi tanımlıyoruz.*/
SqlDataAdapter daPersonel; /* Personel tablosundaki verileri, dtPersonel tablosuna yüklemek için SqlDataAdapter nesnemizi tanımlıyoruz.*/
DataTable dtPersonel; /* Personel tablosundaki verilerin bellek görüntüsünü referans edecek DataTable nesnemizi tanımlıyoruz.*/
DataTable dtDegisiklikler; /* dtPersonel, DataTable nesnesi için AcceptChanges metodu uygulanana kadar meydana gelen değişikliklerin kümesini referans edecek DataTable nesnemizi tanımlıyoruz.*/
/* Kullanıcı bu butona bastığında, Sql sunucumuzdaki Personel tablosunun tüm satırları, dtPersonel DataTable nesnesinin bellekte işaret ettiği alana yüklenecek ve bu veriler DataGrid kontrolüne bağlanacak.*/
private void btnYukle_Click(object sender, System.EventArgs e)
{
```

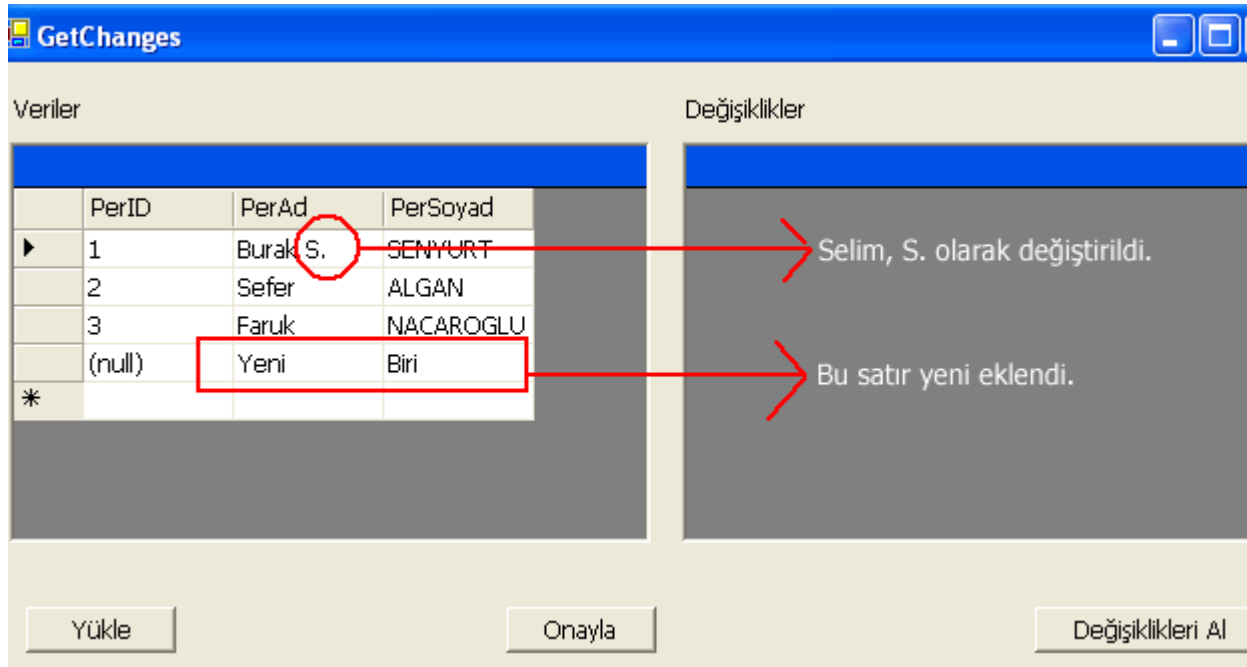


```

conNorthwind=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=sspi");
daPersonel=new SqlDataAdapter("Select * From Personel",conNorthwind);
dtPersonel=new DataTable();
daPersonel.Fill(dtPersonel);
dgVeriler.DataSource=dtPersonel;
}
private void btnOnayla_Click(object sender, System.EventArgs e)
{
    dtPersonel.AcceptChanges(); /* dtPersonel tablosunda meydana gelen değişiklikleri onaylıyoruz.*/
}
private void btnDegisiklikler_Click(object sender, System.EventArgs e)
{
    dtDegisiklikler=new DataTable();
    dtDegisiklikler=dtPersonel.GetChanges(); /* GetChanges metodu ile dtPersonel DataTable nesnesinin işaret ettiği bellek bölgesinde yer alan veri kümesinde meydana gelen değişiklikleri, dtDegisiklikler DataTable nesnesinin bellekte referans ettiği bölgeye alıyoruz. */
    dgDegisiklikler.DataSource=dtDegisiklikler; /* Bu değişiklikleri DataGrid kontrolünde gösteriyoruz. */
}

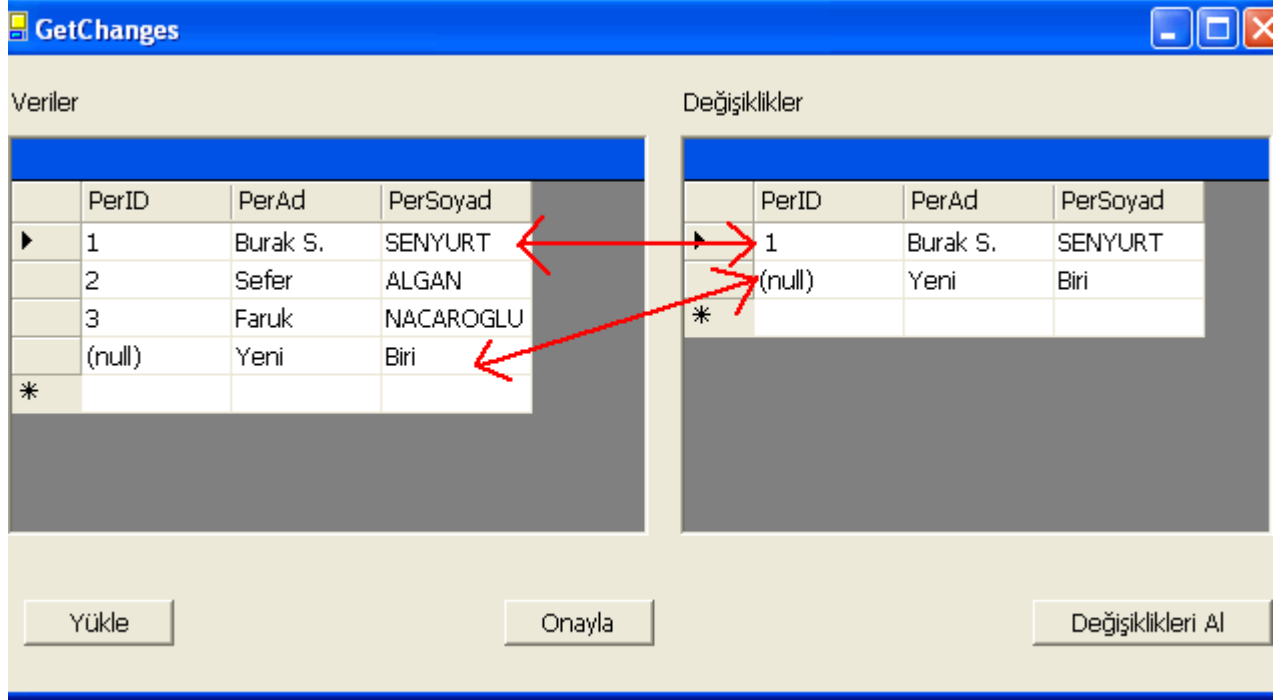
```

Şimdi programımızı çalıştıralım ve tablomuzdaki veriler üzerinde değişiklik yapalım. Örneğin yeni bir satır girelim ve bir satır üzerinde de değişiklik yapalım.



Şekil 2. Yeni bir satır ekleyip bir satır üzerinde değişiklik yaptık.

Şimdi Değişiklikleri Al başlıklı butona tıkladığımızda, GetChanges metodu devreye girecek ve yaptığımız bu değişikliklerin meydana geldiği satırlar aşağıdaki gibi , dtDegisiklikler DataTable nesnesini bağladığımız DataGrid kontrolünde görünecek.



Şekil 3. Yapılan Değişikliklerin Görüntüsü.

Şimdi bu noktadan sonra, dtDegisiklikler isimli DataTable nesnesi üzerinden, SqlDataAdapter nesnesini Update metodunu kullanmak daha akılcıca bir yaklaşım olacaktır. Diğer yandan, GetChanges metodunun bu kullanımı, DataTable(DataSet) de meydana gelen her tür değişikliği almaktadır. Ancak dilersek, sadece yeni eklenen kayıtları ya da sadece değişiklik yapılan kayıtlarda elde edebiliriz. Bunu gerçekleştirmek için, GetChanges metodunun DataRowState numaralandırıcısı türünden parametre aldığı versiyonunu kullanırız. Bu parametre her bir satırın yani DataRow nesnesinin durumunu belirtmektedir ve alabileceği değerler aşağıdaki tabloda verilmiştir.

DataRowState Değeri	Açıklama
Added	DataRowCollection koleksiyonuna yeni bir satır yani DataRow eklenmiş ve AcceptChanges metodu henüz çağırılmamıştır.
Deleted	Delete metodu ile bir satır silinmiştir.
Detached	Yeni bir satır, DataRowCollection için oluşturulmuş ancak henüz Add metodu ile bu koleksiyona dolayısıyla DataTable'a eklenmemiştir.

Modified	Satırda değişiklikler yapılmış ve henüz AcceptChanges metodu çağırılmamıştır.
Unchanged	Son AcceptChanges çağırısından bu yana, satırda herhangi bir değişiklik olmamıştır.

Tablo 1. DataRowState Numaralandırıcısının Değerleri

Şimdi GetChanges metodunun, DataRowState numaralandırıcısı kullanılarak nasıl çalıştığını incelemeye çalışalım. Bunun için aşağıdaki örnek formu tasarlayalım. Bu kez programımızda, değişiklik olan satırları alacak ve bunların durumlarını da gösterecek bir uygulama oluşturacağız.

Şekil 4. Yeni Formumuz.

Formumuza bir ComboBox ekledik. Kullanıcı bu ComboBox'tan DataRowState değerini seçecek ve GetChanges metodumuz buna göre çalışacak. ComboBox'ımızın öğeleri ise şunlar olucak;

Şekil 5. ComboBox öğelerimiz.

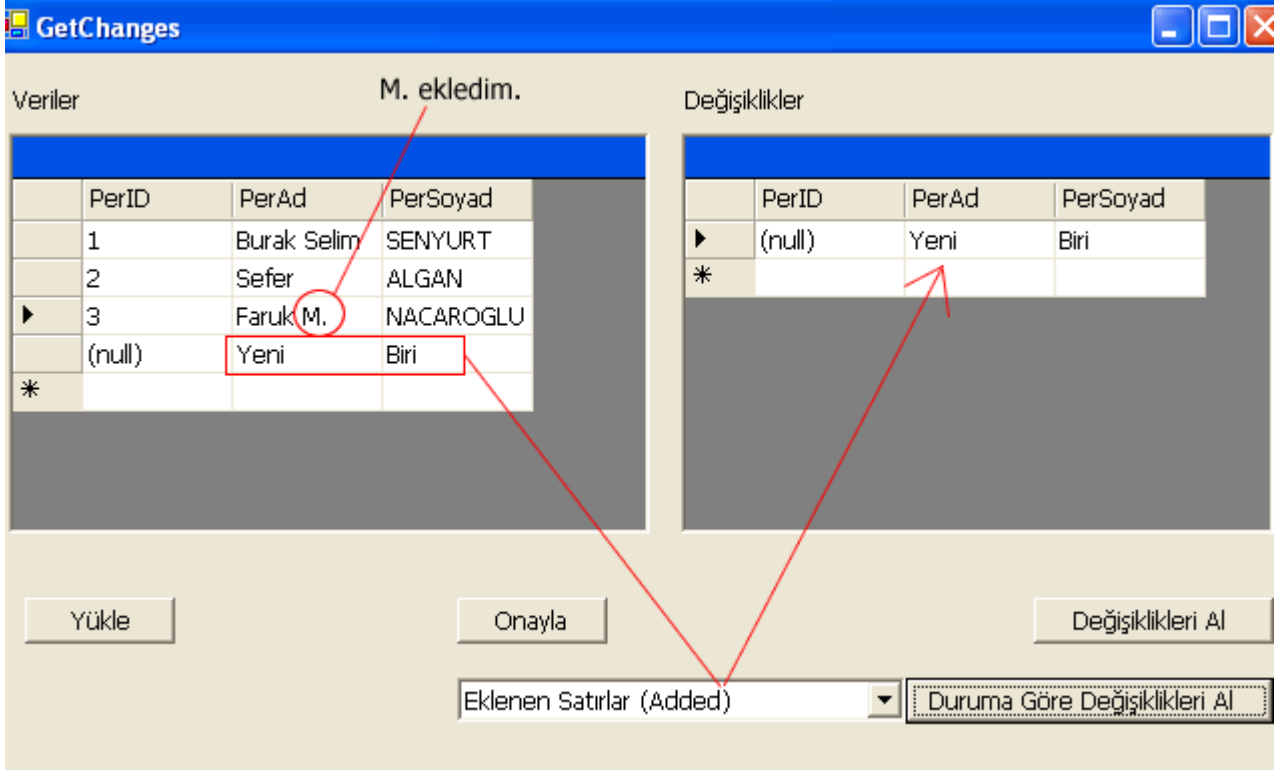
Uygulamamıza sadece, Duruma Göre Değişiklikleri Al başlıklı butonumuzun kodlarını ekleyeceğiz.

```

private void btDurumaGoreDegisiklikler_Click(object sender, System.EventArgs e)
{
    dtDegisiklikler=new DataTable();
    if(cmbRowState.SelectedIndex==0)
    {
        dtDegisiklikler=dtPersonel.GetChanges(DataRowState.Detached); /* Yeni açılan ancak henüz DataRowCollection'a
eklenmeyen satırlar.*/
        dgDegisiklikler.DataSource=dtDegisiklikler;
    }
    else if(cmbRowState.SelectedIndex==1)
    {
        dtDegisiklikler=dtPersonel.GetChanges(DataRowState.Added); /* DataRowCollection'a yeni eklenen satırlar.*/
        dgDegisiklikler.DataSource=dtDegisiklikler;
    }
    else if(cmbRowState.SelectedIndex==2)
    {
        dtDegisiklikler=dtPersonel.GetChanges(DataRowState.Deleted); /* DataTable'dan silinen satırlar.*/
        dgDegisiklikler.DataSource=dtDegisiklikler;
    }
    else if(cmbRowState.SelectedIndex==3)
    {
        dtDegisiklikler=dtPersonel.GetChanges(DataRowState.Modified); /* Değişikliğe uğrayan satırlar.*/
        dgDegisiklikler.DataSource=dtDegisiklikler;
    }
    else if(cmbRowState.SelectedIndex==4)
    {
        dtDegisiklikler=dtPersonel.GetChanges(DataRowState.Unchanged); /* Son AcceptChanges'den sonra değişikliğe uğramamış
satırlar.*/
        dgDegisiklikler.DataSource=dtDegisiklikler;
    }
}

```

Şimdi uygulamamızı çalıştıralım ve deneyelim. Tablomuzda yine birtakım değişiklikler yapalım. Örneğin satırlar ekleyelim, satırları güncelleyelim (Satır eklemek ve satır güncellemek en çok yaptığımız işlemlerdir dikkat ederseniz). Sonrada ComboBox kontrolümüzden istediğimiz durumu seçip elde ettiğimiz sonucu görelim. Örneğin ben yeni bir satır ekledim ve bir satır üzerinde değişiklik yaptım. Daha sonra sadece yeni eklenen satırları görmek için ComboBox kontrolünde Yeni Eklenen Satırlar(Added) seçeneğini seçip düğmeye bastım. Bu durumda if koşumuz durumu değerlendirir ve GetChanges metodunu DataRowState.Added parametresi ile uygular. Sonuç olarak, değişiklik yaptığım satır görünmez sadece yeni eklenen satır dtDegisiklikler tablosuna alınır.



Şekil 6. DataRowState.Added Sonrası.

Bu noktadan sonra artık bir veri tablosunu güncellerken, GetChanges yaklaşımını kullanarak, örneğin sadece yeni eklenen satırların veri kaynağına gönderilmesini sağlamış oluruz. Buda bize daha hızlı ve rahat bir ağ trafiği sağlayacaktır. Bu durum özellikle web servisleri için çok idealdir. Uzak sunuculardan ilgili verileri bilgisayarına bağlantısız olarak işlemek için alan bir istemci uygulama, veri kümesinin tamamını geri göndermek yerine, sadece yeni eklenen veya güncellenen satırları temsil eden bir veri kümesini(dataTable veya DataSet) geri göndererek sınırlı internet kapasitesi için en uygun başarıyı elde edebilir.

Geldik bir makalemizin daha sonuna. Hepinize mutlu günler dilerim.

Stored Procedureler ve ParameterDirection Numaralandırıcısı

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, Sql sunucularında yazdığımız Stored Procedure'lere ilişkin parametreleri incelemeye çalışacağız. Stored Procedure'ler ile ilgili daha önceki makalelerimizde, uygulamamızdan bu procedure'lere nasıl parametre aktarılacağını incelemiştik. Parametre aktarımında yaptığımız işlem, SqlCommand nesnesimizin parametre koleksiyonuna, Stored Procedure içinde tanımladığımız parametrenin eklenmesiydi. Bunun için, SqlCommand sınıfının Parameters koleksiyonuna Add metodunu kullanarak SqlParameter sınıfı türünden bir nesne ekliyorduk. Bu parametrelere program içerisinde ilgili değerleri aktararak, bu değerlerin Stored Procedure içine aktarılmasına imkan sağlıyorduk.

Bugünkü makalemizde ise, bir Stored Procedure'den programımıza nasıl değer(ler) döndürebileceğimizi inceleyeceğiz. Dikkat ederseniz, bir Stored Procedure'e program içinden parametre aktarabileceğimiz gibi, Stored Procedure'dende programımıza değerler aktarabildiğimizden bahsediyoruz. Dolayısıyla parametrelerin bir takım farklı davranışlar sergileyebilmesi söz konusu.

SqlParameter sınıfı, parametrelerin davranışlarını yada başka bir deyişle hangi yöne doğru hareket edeceklerini belirten bir özellik içermektedir. Bu özellik Direction özelliğidir ve C# prototipi aşağıdaki gibidir.

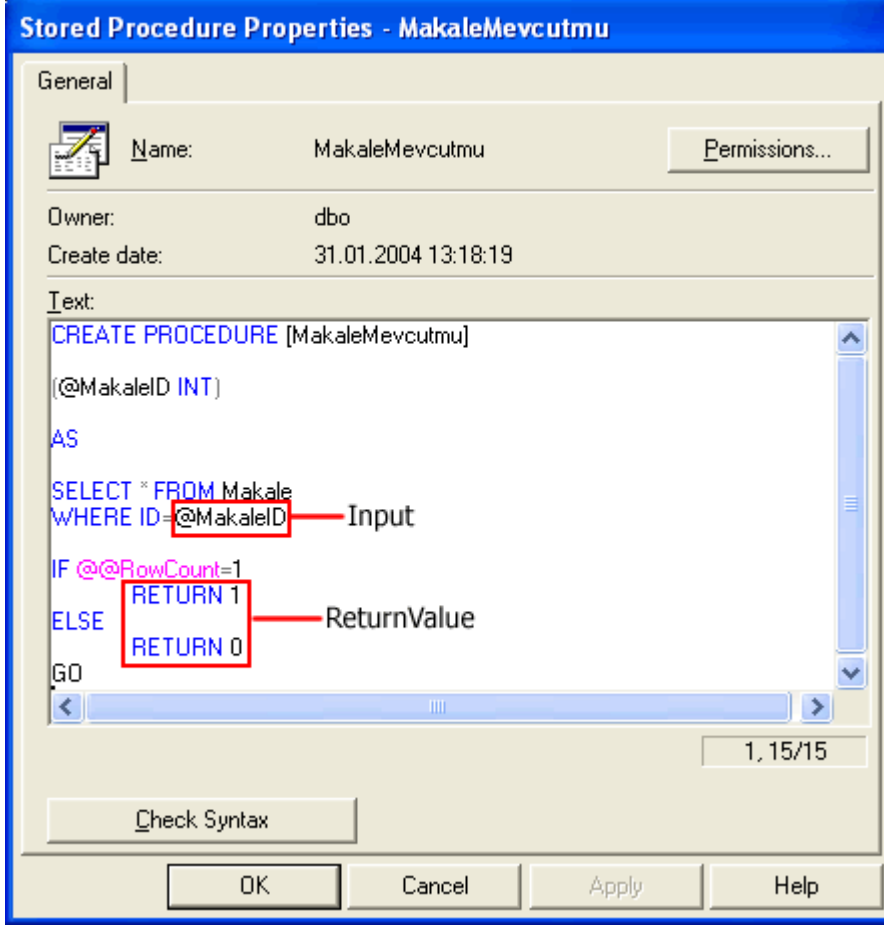
```
public virtual ParameterDirection Direction {get; set;}
```

Direction özelliği, ParameterDirection numaralandırıcısı tipinden değerler almaktadır. Bu değerlerin açıklaması aşağıdaki tabloda yer almaktadır.

Direction Değeri	Açıklama
Input	Bir SqlParameter'sinin varsayılan değeri budur. Program içinden Stored Procedure'e değerler gönderileceği zaman, SqlParameter nesnesinin Direction özelliği Input olarak kullanılır. Yani parametre değerinin yönü Stored Procedure'e doğrudur.
Output	Stored Procedure'den programımıza doğru değer aktarımı söz konusu ise SqlParameter nesnesinin Direction değeri Output yapılır. Bu, parametre yönünün, Stored Procedure'den programımıza doğru olduğunu göstermektedir.
ReturnValue	Bazen bir Stored Procedure'ün çalışması sonucunu değerlendirmek isteyebiliriz. Bu durumda özellikle Stored Procedure'den Return anahtar sözcüğü ile döndürülen değerler için kullanılan parametrelerin Direction değeri ReturnValue olarak belirlenir. Bu tip bir parametreyi, bir fonksiyonun geri döndürdüğü değeri işaret eden bir parametre olarak düşünebiliriz.
InputOutput	Bu durumda parametremiz hem Input hemde Output yeteneklerine sahip olacaktır.

Tablo 1.ParameterDirection Numaralandırıcısının Değerleri.

Bugünkü makalemizde ağırlıklı olarak ReturnValue ve Output durumlarını incelemeye çalışacağız. Dilerseniz işe ReturnValue ile başlayalım. Bu örnekler için, Makaleler ile ilgili bilgilere sahip olan bir tablo kullanacağız. Şimdi sql sunucumuzda, bu tablo için aşağıdaki Stored Procedure nesnesini oluşturalım.



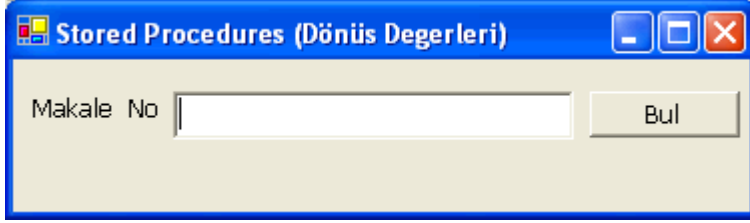
Şekil 1. MakaleMevcutmu Stored Procedure'ümüz.

Bu Stored Procedure, programdan @MakaleID isimli bir parametre alıyor. Bu parametreyi, SELECT sorgusuna geçiriyor ve Makale tablosunda ID alanı, @MakaleID parametresinin değerine eşit olan satırı seçiyor. Burada kullanılan @MakaleID parametresinin değeri program içinden belirlenecektir. Dolayısıyla bu parametremizin yönü, programımızdan Stored Procedure'ümüze doğrudur. Bu nedenle, programımızda bu parametre tanımlandığında Direction değeri Input olmalıdır. Ancak bunu program içinde belirtmeyecek yani SqlParameter nesnesinin Direction özelliğine açıkça ParameterDirection.Input değerini atamayacağız. Çünkü bir SqlParameter nesnesinin Direction özelliğinin varsayılan değeri budur.

Gelelim RETURN kısmına. Burada @@RowCount isimli sql anahtar kelimesi kullanıyoruz. Bu anahtar kelime Stored Procedure çalıştırdıktan sonra, Select sorgusu sonucu dönen satır sayısını vermektedir. If koşulumuz ile, procedure içinden, bu select sorgusu sonucunun değerine bakıyoruz. Eğer PrimaryKey olan ID alanımızda, @MakaleID parametresine atadığımız değer mevcutsa, @@RowCount, 1 değerini döndürecek. Daha sonra, RETURN anahtar kelimesini kullanarak, Stored Procedure'den programa değer döndürüyoruz. İşte bu değerler, programımızdaki SqlParameter nesnesi için, Direction özelliğinin ParameterDirection.ReturnValue olmasını gerektirir. Örneğimizi tamamladığımızda bu durumu daha iyi anlayacağınıza inanıyorum.

Şimdi aşağıdaki Form tasarımı yapalım ve kodlarımızı yazalım. Program basit bir şekilde, bu Stored Procedure'e kullanıcının girdiği Makale numarasını gönderecek ve sonuçta bu makalenin var olup olmadığını bize belirtecek. Bu arada şunuda belirtmekte fayda var. Bu tip bir işlemi elbette, SqlCommand nesnesine, bir Select sorgusu girerek de gerçekleştirebilirdik. Ancak Stored

Procedure'lerin en önemli özelliklerinin, derlenmiş sql nesneleri oldukları için, sağladıkları performans ve hız artışı kazanımları olduğunda görmezden gelemeyiz. Bununla birlikte güvenlik açısından daha verimlidirler. Özellikle web uygulamaları için. Şimdi Formumuzu tasarlayalım. Bunun için basit bir windows uygulaması geliştireceğiz.



Şekil 2. Form Tasarımımız.

```
private void button1_Click(object sender, System.EventArgs e)
{
    conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi"); /* Sql sunucumuza
    olan bağlantımızı gerçekleştiriyoruz. */

    SqlCommand cmd=new SqlCommand("MakaleMevcutmu",conFriends); /* SqlCommand nesnemizi oluşturuyoruz. SqlCommand
    nesnemize, Stored Procedure'ümüzün adını parametre olarak veriyoruz. */

    cmd.CommandType=CommandType.StoredProcedure; /* SqlCommand nesnemiz bir Stored Procedure çalıştıracağı için
    CommandType özelliği CommandType.StoredProcedure olarak belirlenir.*/

    /* Burada, Stored Procedure'ümüzden Return ile dönen değeri işaret edecek SqlParameter nesnemizi ,SqlCommand nesnemizin
    Parameters koleksiyonuna Add metodu ile ekliyoruz. Return anahtar sözcüğü ile geri dönen değeri işaret edecek paramterenin adı
    herhangi bir isim olabilir. Ancak her sql parametresinde olduğu gibi başında @ işaretini kullanmayı unutmamalıyız.*/
    cmd.Parameters.Add("@DonenDeger",SqlDbType.Int);
    cmd.Parameters["@DonenDeger"].Direction=ParameterDirection.ReturnValue; /* Parametrenin, Stored Procedure'den,
    programa doğru olduğunu ve Return anahtar sözcüğü ile geriye dönen bir değeri işaret ettiğini belirtmek için, Direction özelliğine,
    ParameterDirection.ReturnValue değerini veriyoruz.*/

    /* Burada programımızda kullanıcının girdiği Makale numarasını, Stored Procedure'ümüze doğru gönderecek SqlParameter
    nesnemizi tanımlanıyoruz. Bu parametre Input tipindedir. Bu nedenle, adının, Stored Procedure'ümüzdeki ile aynı olmasına dikkat
    etmeliyiz. */
    cmd.Parameters.Add("@MakaleID",SqlDbType.Int);
    cmd.Parameters["@MakaleID"].Value=txtMakaleNo.Text; /* Parametremizin değeri veriliyor.*/

    /* Güvenli bloğumuzda, öncelikle sql sunucumuza olan bağlantımızı SqlConnection yardımıyla açıyoruz ve ardından
    SqlCommand nesnemizin referans ettiği Stored Procedure nesnemizi çalıştırıyoruz.*/
    try
    {
        conFriends.Open();
        cmd.ExecuteNonQuery();
        int Sonuc;
        /* Stored Procedure'ümüzden Return anahtar sözcüğü ile dönen değeri @DonenDeger SqlParameter nesnesi ile alıyor ve
```

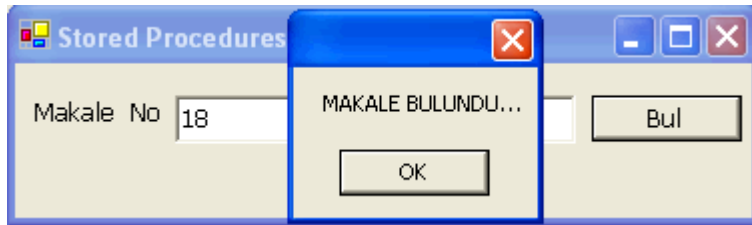

sonuc ismindeki integer tipteki değişkenimize atıyoruz. SqlCommand nesnesinin, Parameters koleksiyonunda yer alan SqlParameter nesnelerinin Value özelliği geriye Object tipinden değer döndürdüğü için, bu değeri integer tipine dönüştürme işlemide uyguladığımıza dikkat edelim. */

```

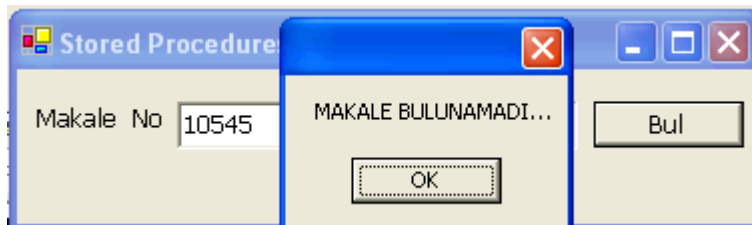
Sonuc=Convert.ToInt32(cmd.Parameters["@DonenDeger"].Value);
/* Burada dönen değeri değerlendirerek kullanıcının girdiği ID'ye sahip bir Makale olup olmadığını belirliyoruz.*/
if(Sonuc==1)
{
    MessageBox.Show("MAKALE BULUNDU...");
}
else if(Sonuc==0)
{
    MessageBox.Show("MAKALE BULUNAMADI...");
}
}
catch(Exception hata)
{
    MessageBox.Show("Hata:"+hata.Message.ToString());
}
finally
{
    conFriends.Close(); /* Bağlantımızı kapatıyoruz. */
}
}

```

Şimdi programımızı çalıştıralım ve bir ID değeri girelim. Bu noktada kullanıcının girdiği ID değeri, @MakaleID isimli SqlParameter nesnemiz yardımıyla, Stored Procedure'ümüze aktarılacak ve Stored Procedure'ün çalışması sonucu geri dönen değerde @DonenDeger isimli SqlParameter nesnesi yardımıyla uygulamamızda değerlendirilecek.

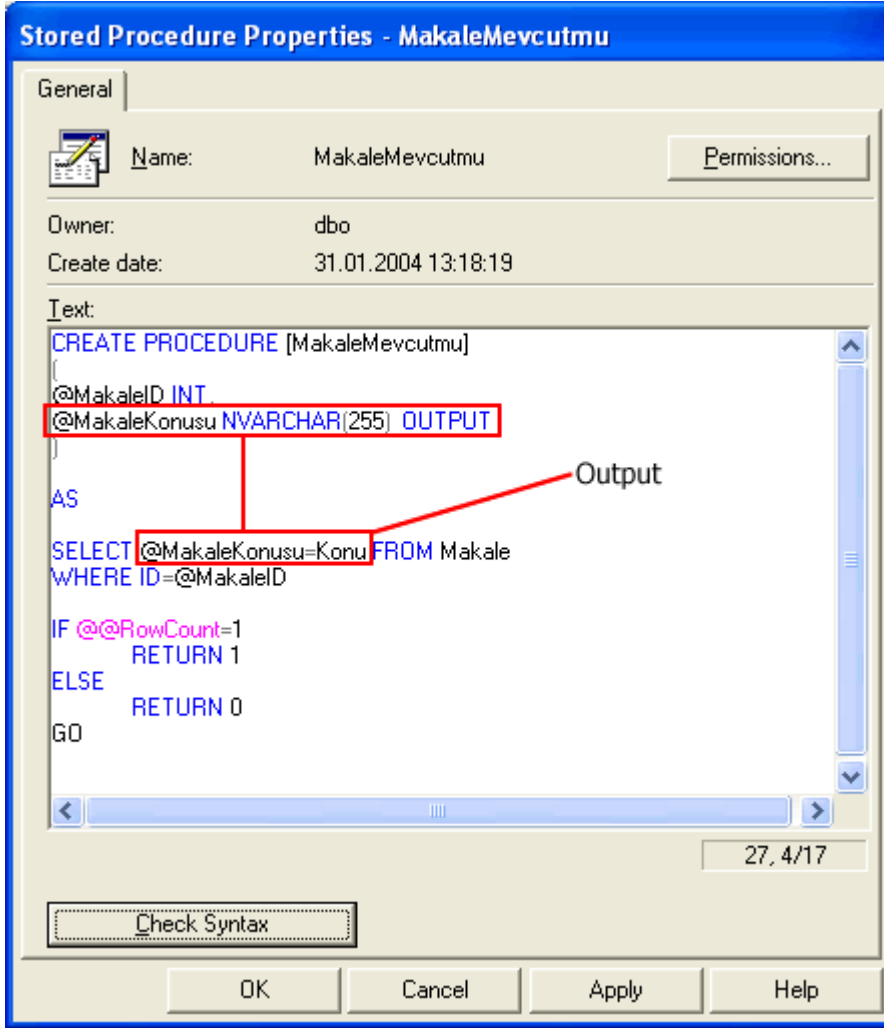


Şekil 3. Programın Çalışması Sonucu. Makale bulunduğunda.



Şekil 4. Makale bulunamadığında.

Elbette bu örnek bize pek bir şey ifade etmiyor. Nitekim makalenin var olup olmadığının farkına vardık o kadar. Ama Makale tablosundaki belli alanlarda görmek istediğimizi varsayalım. İşte bu, Output tipindeki SqlParameter nesnelerini kullanmak için güzel bir fırsat. Şimdi bu Stored Procedure nesnemizin sql kodunu biraz değiştireceğiz.



Şekil 5. Output Tipi Sql Parametreleri.

Burada Select sorgusundaki @MakaleKonusu=Konu ifadesine dikkatinizi çekmek isterim. Eğer @MakaleID parametresinin değerinde bir Makale satırı var ise, bu satırın Konu alanının değerini, @MakaleKonusu isimli parametreye aktarıyoruz. İşte bu parametre, programımıza doğru dönen Output tipinde bir parametredir. Diğer yandan Output parametrelerini kullanırken, sql yazımı içindede bu parametre değişkeninin OUTPUT anahtar sözcüğü ile belirtilmesi gerekmektedir. Yeni duruma göre program kodlarımız aşağıdaki gibi olmalıdır.

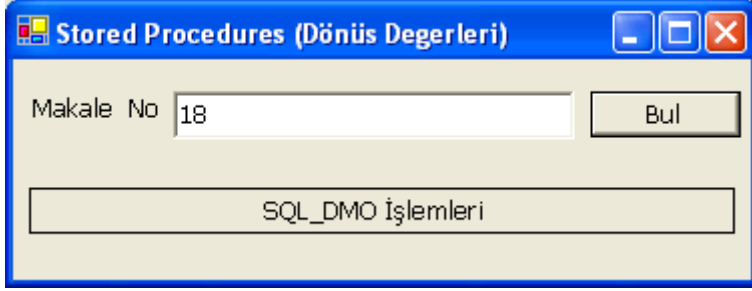
```
private void button1_Click(object sender, System.EventArgs e)
{
    conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
```

```

SqlCommand cmd=new SqlCommand("MakaleMevcutmu",conFriends);
cmd.CommandType=CommandType.StoredProcedure;
cmd.Parameters.Add("@DonenDeger",SqlDbType.Int);
cmd.Parameters["@DonenDeger"].Direction=ParameterDirection.ReturnValue;
cmd.Parameters.Add("@MakaleKonusu",SqlDbType.NVarChar,255);
cmd.Parameters["@MakaleKonusu"].Direction=ParameterDirection.Output;
cmd.Parameters.Add("@MakaleID",SqlDbType.Int);
cmd.Parameters["@MakaleID"].Value=txtMakaleNo.Text;
try
{
    conFriends.Open();
    cmd.ExecuteNonQuery();
    int Sonuc;
    string MakaleAdi;
    Sonuc=Convert.ToInt32(cmd.Parameters["@DonenDeger"].Value);
    MakaleAdi=cmd.Parameters["@MakaleKonusu"].Value.ToString();
    if(Sonuc==1)
    {
        MessageBox.Show("MAKALE BULUNDU...");
        lblMakaleKonusu.Text=MakaleAdi;
    }
    else if(Sonuc==0)
    {
        MessageBox.Show("MAKALE BULUNAMADI...");
    }
}
catch(Exception hata)
{
    MessageBox.Show("Hata:"+hata.Message.ToString());
}
finally
{
    conFriends.Close();
}
}

```

Şimdi uygulamamızı çalıştıralım. Aşağıdaki sonucu elde ederiz.



Şekil 6. Makalemizin Konu isimli alanının değeri döndürüldü.

Değerli okurlarım. Geldik bir makalemizin daha sonuna. Umarım yararlı bir makale olmuştur. Hepinize mutlu günler dilerim.

Strongly Typed DataSet - 1 (Kuvvetle Türlendirilmiş Veri Kümeleri)

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde kuvvetle türlendirilmiş veri kümelerinin ne olduğunu ve nasıl oluşturulduklarını incelemeye çalışacağız. Kuvvetle türlendirilmiş veri kümelerini tanımlamadan önce, aşağıdaki kod satırının incelemekle işe başlayalım.

```
textBox1.Text=dsMakale1.Tables["Makale"].Rows[3]["Konu"].ToString();
```

Bu satır ile, dsMakale isimli dataSet nesnemizin bellekte işaret ettiği veri bölgesinde yer alan DataTable'lardan Makale tablosuna işaret edenin, 4ncü satırındaki Konu alanının değeri alınarak, TextBox kontrolümüzün text özelliğine atanmaktadır. Şimdide aşağıdaki kod satırını ele alalım.

```
textBox1.Text=rsMakale.Fields("Konu").Value
```

Bu ifade eski dostumuz ADO daki rsMakale isimli recordSet'i kullanarak, Konu isimli alanın değerine ulaşmıştır. Dikkat edecek olursanız bu iki ifade arasında uzunluk açısından belirgin bir fark vardır. İkinci yazım daha kolaydır. Zaten bu nedenle, ADO.NET'i öğrenen programcıların ilk başta en çok karşılaştıkları zorluk, bu kod yazımının uzunluğu olmuştur. Bununla birlikte, ADO.NET' in XML tabanlı bir mimariye sahip olması, karşımıza Kuvvetle Türlendirilmiş Veri Kümelerini çıkarmaktadır. Microsfot.NET mimarları, programlarımızda aşağıdakine benzer daha kısa ifadelerin kullanılabilmesi amacıyla, Kuvvetle Güçlendirilmiş Veri Kümeleri kavramını ADO.NET'e yerleştirmiştir.

```
textBox1.Text=dsTypedMakale.Makale[3].Konu.ToString();
```

Bu ifade ilk yazdığımız ifadeye göre çok daha kısadır. Peki bu nasıl sağlanmıştır. dsTypedMakale isimli DataSet nesnemiz aslında Kuvvetle Türlendirilmiş Veri Kümemizin ta kendisidir. Bu noktada Kuvvetle Türlendirilmiş Veri Kümesi'nin ne olduğunu tanımlayabiliriz.

Bir Strongly Typed DataSet (Kuvvetle Türlendirilmiş Veri Kümesi), DataSet sınıfından türetilmiş, programcı tarafından belirtilen bir xml schema sınıfını baz alan, veri bağlantılarının özelleştirilip yukarıdaki gibi kısa yazımlar ile erişimlere imkan sağlayan, özelleştirilmiş bir DataSet sınıfıdır.

Bu geniş kavramı anlamak için elbette en kolay yol örneklendirmek olacak. Ancak bundan önce Kuvvetle Türlendirilmiş Veri Kümemizin nasıl oluşturacağımızı görelim. Bunun için elimizde iki yol var. Yollardan birisi, komut satırından XSD.exe XML Schema Defination Tool'unu kullanmak. Bu yol biraz daha zahmetli olmakla birlikte Visual Studio.NET'e olan gereksinimi kaldırdığı için zaman zaman tercih edilir. Diğer yolumuz ise Kuvvetle Türlendirilmiş Veri Kümemizi Visual Studio.NET ortamında oluşturmak. Yollardan hangisini seçersek seçelim, Kuvvetle Türlendirilmiş Veri Kümemizin oluşturulması için bize mutlaka bir xml schema dosyası (xsd uzantılı) gerekiyor. Çünkü Kuvvetle Türlendirilmiş Veri Kümemiz, bir xml schema dosyası baz alınarak oluşturulmaktadır. Şimdi dilerseniz komut satırı yardımıyla bir Kuvvetle Türlendirilmiş Veri Kümesinin nasıl oluşturulacağını inceleyelim. Bunun için öncelikle biraz kod yazmamız gerekecek. İzleyen kodlar ile, bir DataSet'i gerekli tablo bilgileri ile yükleyecek ve daha sonra bu DataSet'e ait xml schema bilgilerini, DataSet sınıfına ait WriteXmlSchema metodu ile bir xsd dosyasına aktaracağız. Örneğin basit olması amacıyla bir console uygulaması oluşturalım.

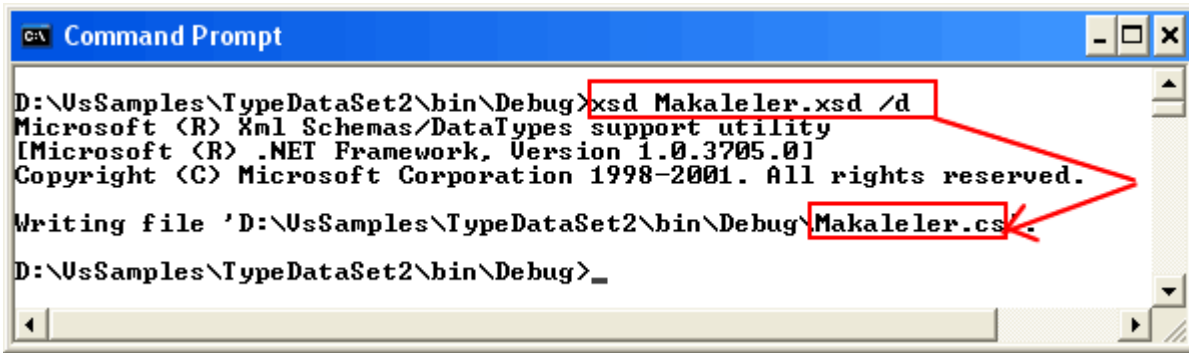
```
using System;
using System.Data;
using System.Data.SqlClient;
namespace TypeDataSet2
{
    class Class1
    {
        static void Main(string[] args)
        {
            SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
            SqlDataAdapter da=new SqlDataAdapter("Select * From Makale",conFriends);
            DataSet dsMakale=new DataSet("Makaleler");
            conFriends.Open();
            da.FillSchema(dsMakale,SchemaType.Source,"Makale");
            conFriends.Close();
            dsMakale.WriteXmlSchema("Makaleler.xsd");
        }
    }
}
```

Yukarıdaki kodları kısaca açıklayalım. Burada Sql sunucumuzda yer alan Friends isimli veritabanına bağlanıyor ve Makale isimli tabloya ait schema bilgilerini DataSet nesnemize yüklüyoruz. Daha sonra ise, DataSet nesnemizin WriteXmlSchema metodu ile, DataSet'in yapısını xml tabanlı schema dosyası olarak (Makaleler.xsd) sisteme kaydediyoruz. İşte Kuvvetle Türlendirilmiş Veri Kümemizi bu xsd uzantılı schema dosyası yardımıyla oluşturacağız. Sistemizde uygulamamızı oluşturduğumuz yerdeki Debug klasörüne baktığımızda, aşağıdaki görüntüyü elde ederiz.



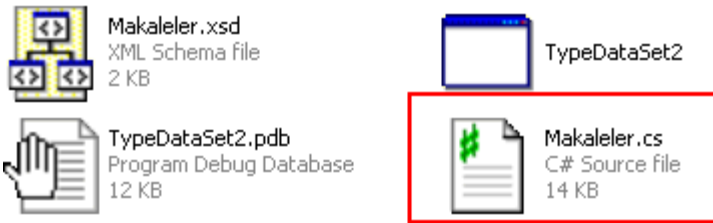
Şekil 1. XML Schema Dosyamız.

Şimdi .NET Framework'ün XSD.exe aracını kullanarak, bu schema dosyasından Kuvvetle Türlendirilmiş Veri Kümemizi temsil edecek sınıfı oluşturalım. Bunun için, komut satırında aşağıdaki satırı yazalım.



Şekil 2. Kuvvetle Türlendirilmiş Veri Kümesi sınıfının oluşturulması.

Burada görüldüğü gibi, xsd schema dosyamızdan, Kuvvetle Türlendirilmiş Veri Kümesi sınıfımız (Makaleler.cs) oluşturulmuştur. xsd araçındaki /d parametresi, sınıfımızın DataSet sınıfından türetilmesini belirtmektedir. Şimdi Debug klasörümüze tekrar bakacak olursak, Kuvvetle Türlendirilmiş Veri Kümesi sınıfımızın oluşturulmuş olduğunu görürüz.



Şekil 3. Kuvvetle Türlendirilmiş Veri Kümesi Sınıfımız.

Peki bu oluşturduğumuz Kuvvetle Türlendirilmiş Veri Kümesi sınıfını uygulamamızda nasıl kullanacağız. Bu oldukça basit. Yeni DataSet nesnemizi, Makaleler.cs sınıfından türeteceğiz. Aşağıdaki kodu inceleyelim.

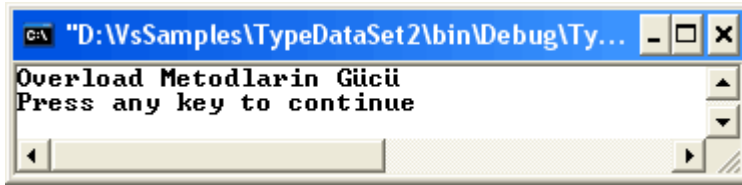
```
using System;
using System.Data;
using System.Data.SqlClient;
namespace TypeDataSet2
{
```

```

class Class1
{
    static void Main(string[] args)
    {
        SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
        SqlDataAdapter da=new SqlDataAdapter("Select * From Makale",conFriends);
        conFriends.Open();
        Makaleler dsTypedMakale=new Makaleler(); /* Kuvvetle Türlendirilmiş Veri Kümesi sınıfımızdan bir DataSet nesnesi
türetiyoruz.*/
        da.Fill(dsTypedMakale.Makale); /* SqlDataAdapter nesnemiz yardımıyla, yeni dataSet'imizdeki Makale isimli tablomuzu
Sql sunucumuzda yer alan Makale isimli tablonun verileri ile yüklüyoruz. */
        Console.WriteLine(dsTypedMakale.Makale[3].Konu.ToString()); /* Yeni DataSet nesnemizdeki Makale tablosunun 4ncü
satırındaki Konu alanının değerine erişiyoruz. */
        conFriends.Close();
    }
}

```

Uygulamamızı çalıştırdığımızda aşağıdaki sonu elde ederiz.



Şekil 4. Uygulamanın sonucu.

Görüldüğü gibi bir Kuvvetle Türlendirilmiş Veri Kümesi oluşturmak ve kullanmak son derece basit. Dilerseniz xsd aracı ile oluşturduğumuz Makaleler.cs isimli Kuvvetle Türlendirilmiş Veri Kümesi sınıfımızı ele alalım. Bu sınıfı oluşturduğunuzda mutlaka incelemenizi tavsiye ederim. Çok uzun bir dosya olduğundan tüm kod satırlarına burada yer vermiyorum. Ancak dikkatimizi çekicek bir kaç üyeyi göstereceğim. Öncelikle sınıf tanımımıza bir göz atalım.

```

public class Makaleler : DataSet
{
    ...
}

```

Daha öncedende söylediğimiz gibi Kuvvetle Türlendirilmiş Veri Kümesi sınıfları DataSet sınıfından türetilmektedir. Yeni dataSet sınıfımız, içereceği tablo, alan, kısıtlama, ilişki gibi bilgileri bir xsd dosyasından almaktaydı. Biz xsd dosyasını oluştururken, DataSet'e Makale isimli tablonun yapısını yüklemiştik. Bu durumda, Kuvvetle Türlendirilmiş Veri Kümesi sınıfımız içinde, Makale isimli tablomuzu belirten yeni bir DataTable üyesi kullanılır.

```
private MakaleDataTable tableMakale;
```

Burada MakaleDataTable, DataTable sınıfının özelleştirilmiş bir haline sunar.

```
protected Makaleler(SerializationInfo info, StreamingContext context)
{
    string strSchema = ((string)(info.GetValue("XmlSchema", typeof(string))));
    if ((strSchema != null))
    {
        DataSet ds = new DataSet();
        ds.ReadXmlSchema(new XmlTextReader(new System.IO.StringReader(strSchema)));
        if ((ds.Tables["Makale"] != null))
        {
            this.Tables.Add(new MakaleDataTable(ds.Tables["Makale"]));
        }
    }
    ...
}
...
public MakaleDataTable Makale
{
    get
    {
        return this.tableMakale;
    }
}
...
```

Görüldüğü gibi MakaleDataTable isminde yeni bir sınıfımız vardır. Bu sınıf Makale isimli tabloyu tüm elemanları ile tanımlar. Bunun için, bu sınıf DataTable sınıfından türetilir. Makale tablosundaki her bir alan bu sınıf içerisinde birer özellik haline gelir. Bununla birlikte bu yeni DataTable sınıfı, alan ekleme, alan silme gibi metodlar ve pek çok olayıda tanımlar. Diğer yandan pek çok DataTable metoduda bu sınıf içinde override edilir.

```
public class MakaleDataTable : DataTable, System.Collections.IEnumerable
{
    private DataColumn columnID;
    private DataColumn columnKonu;
    private DataColumn columnTarih;
    private DataColumn columnAdres;
    ...
}
```

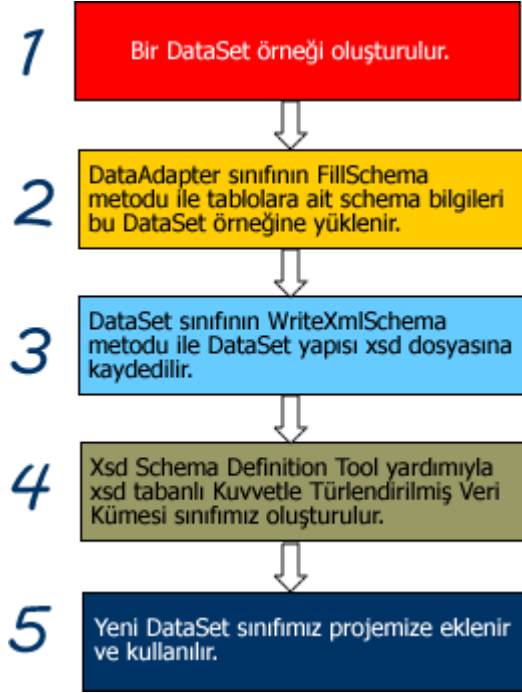


```

public int Count
{
    get
    {
        return this.Rows.Count;
    }
}
internal DataColumn IDColumn
{
    get
    {
        return this.columnID;
    }
}
public MakaleRow this[int index]
{
    get
    {
        return ((MakaleRow)(this.Rows[index]));
    }
}
public event MakaleRowChangeEventHandler MakaleRowChanged;
...
public void AddMakaleRow(MakaleRow row)
{
    this.Rows.Add(row);
}
public MakaleRow AddMakaleRow(string Konu, System.DateTime Tarih, string Adres)
{
    MakaleRow rowMakaleRow = ((MakaleRow)(this.NewRow()));
    rowMakaleRow.ItemArray = new object[] { null, Konu, Tarih, Adres };
    this.Rows.Add(rowMakaleRow);
    return rowMakaleRow;
}
...
}

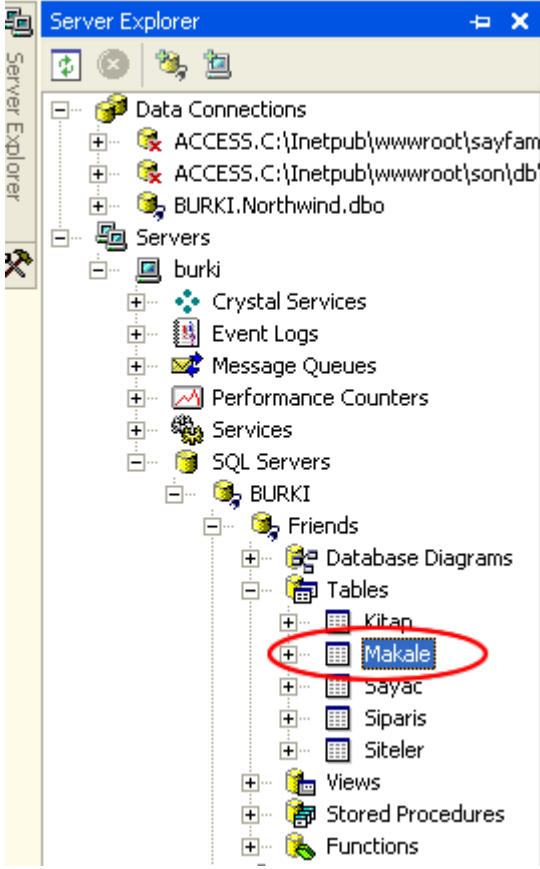
```

Buraya kadar , komut satırı yardımıyla ve kod yazarak bir Kuvvetle Türlendirilmiş Veri Kümesinin nasıl oluşturulacağını gördük. Bu teknikteki adımları gözden geçirmek gerekirse izlememiz gereken yol şöyle olacaktır.



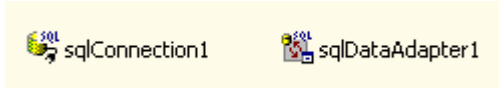
Şekil 5. Kuvvetle Türlendirilmiş Veri Kümesi Oluşturulma Aşamaları

Şimdide, Kuvvetle Türlendirilmiş Veri Kümesi sınıfını, Visual Studio.NET ortamında nasıl geliştireceğimizi görelim. Bu daha kolay bir yoldur. Öncelikle Visual Studio.Net ortamında bir Windows Uygulaması açalım. Daha sonra, Server Explorer penceresinden Makale isimli tablomuzu Formumuza sürükleyelim.



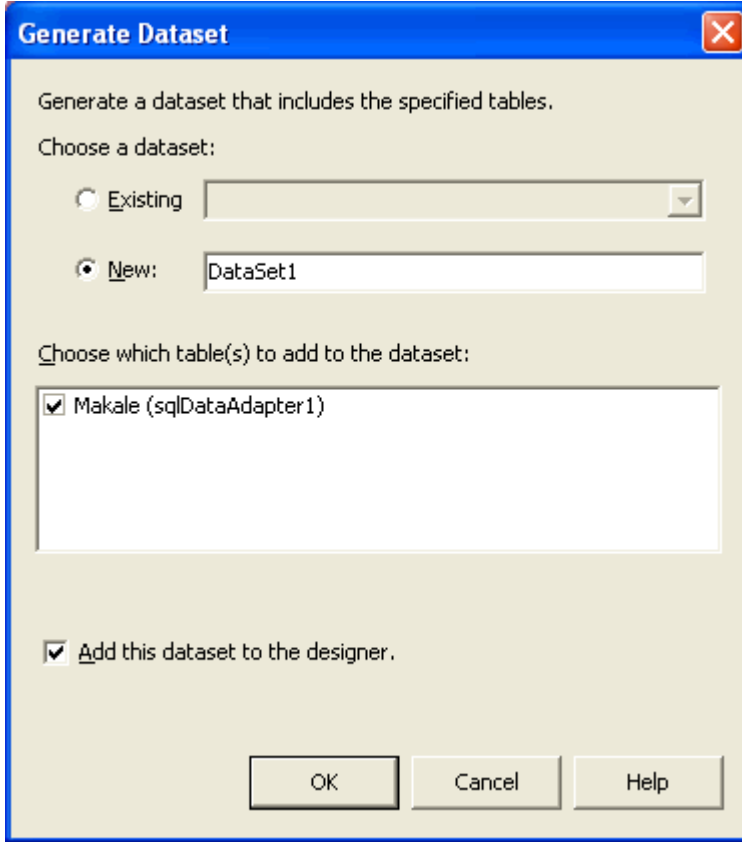
Şekil 6. Server Explorer ile Makale Tablosunun Forma Alınması

Bu durumda formumuzda otomatik olarak SqlConnection ve SqlDataAdapter nesnelerimiz oluşur.



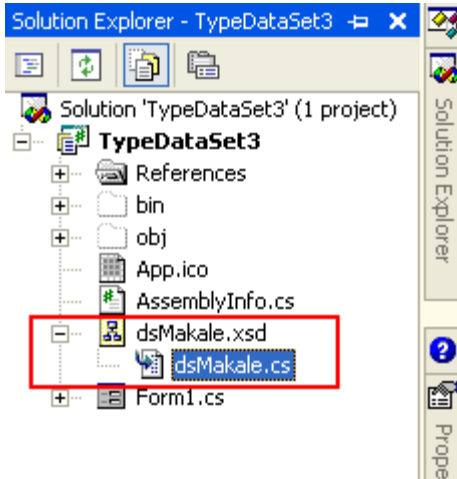
Şekil 7. SqlConnection ve SqlDataAdapter otomatik olarak oluşturulur.

Şimdi SqlDataAdapter nesnemizin özelliklerinden Generate DataSet'e tıkladığımızda karşımıza aşağıdaki pencere çıkacaktır.



Şekil 8. DataSet oluşturuluyor.

Burada DataSet nesnemiz otomatik olarak oluşturulacaktır. DataSet'imize dsMakale adını verelim ve OK başlıklı butona basalım. Şimdi Solution Explorer'da Show All Files seçeneğine tıklarsak, Kuvvetle Türlendirilmiş Veri Kümesi sınıfımızın oluşturulduğunu görürüz.

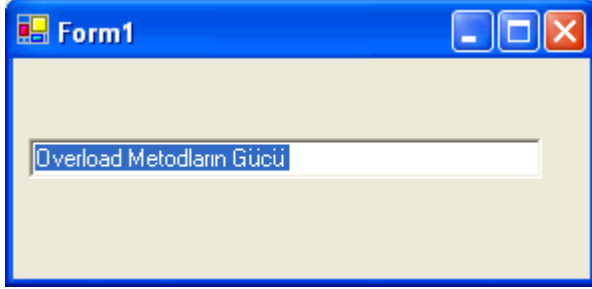


Şekil 9. Kuvvetle Türlendirilmiş Veri Kümesi

Artık uygulamamızda bu sınıfı kullanabiliriz. İşte örnek kod satırlarımız.

```
private void Form1_Load(object sender, System.EventArgs e)
{
    dsMakale mk=new dsMakale();
    sqlDataAdapter1.Fill(mk.Makale);
    textBox1.Text=mk.Makale[3].Konu.ToString();
}
```

Uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 10. Uygulamanın Sonucu.

Bu makalemizde, Kuvvetle Türlendirilmiş Veri Kümelerinin ne olduğunu, nasıl oluşturulacağını ve nasıl kullanılacağını gördük. İzleyen makalemizde bu konuya devam edecek ve Satır Ekleme, Satır Düzenleme, Satır Silme gibi işlemlerin nasıl yapılacağını inceleyeceğiz. Hepinize mutlu günler dilerim.

Strongly Typed DataSet - 2 (Kuvvetle Türlendirilmiş Veri Kümeleri)

Değerli Okurlarım, Merhabalar.

Bir önceki makalemizde, Kuvvetle Türlendirilmiş Veri Kümelerinin ne olduğunu ve nasıl oluşturulduğunu incelemiştik. Bu makalemizde ise, bir türlendirilmiş veri kümesi yardımıyla satır ekleme, arama, düzenleme ve silme gibi işlemlerin nasıl yapılacağını inceleyeceğiz. Bu amaçla işe basit bir windows uygulaması ile başlıyoruz. Bu uygulamamızda kolaylık olması açısından Kuvvetle Türlendirilmiş Veri Kümemizi, Visual Studio.NET ortamında oluşturdum. Uygulamamızda, Makale isimli sql tablomuzu kullanacağız. Uygulamamızın formu izleyen şekildeki gibi olacak.

Şekil 1. Form Tasarımımız.

Kullanıcı, Ekle başlıklı butona tıkladığında, textBox kontrollerine girmiş olduğu değerlerden oluşan yeni satırımız, dataTable'ımıza eklenmiş olacak. Bu işlemi gerçekleştiren kodlar aşağıda verilmiştir.

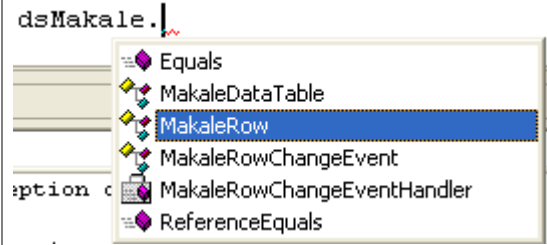
```
private void btnEkle_Click(object sender, System.EventArgs e)
{
    dsMakale.MakaleRow dr; /* Yeni bir satır tanımlanıyor. MakaleRow bir DataRow tipidir ve bizim dsMakale isimli Kuvvetle
    Türlendirilmiş Veri Kümesi sınıfımızda yer almaktadır.*/
    dr=mk.Makale.NewMakaleRow(); /* MakalNewRow ile dr isimli MakaleRow nesnemizin, Makale isimli DataTable'ımızda yeni ve
    boş bir satıra referans etmesi sağlanıyor. */
    dr.Konu=txtKonu.Text; /* Veriler yeni satırımızın ilgili alanlarına yerleştiriliyor. */
    dr.Tarih=Convert.ToDateTime(txtTarih.Text);
    dr.Adres=txtAdres.Text;
    mk.Makale.AddMakaleRow(dr); /* Son olarak AddMakaleRow metodu ile oluşturulan yeni satır dataTable'ımıza ekleniyor.*/
}
```

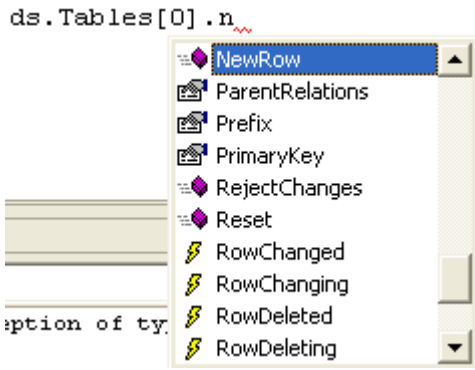
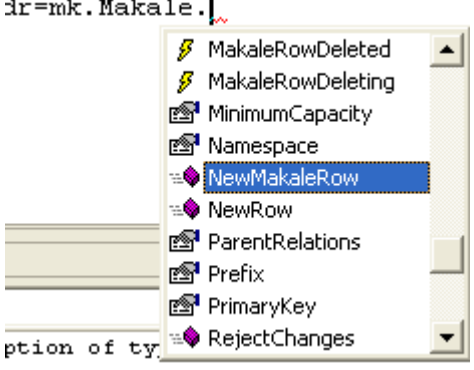
Bu teknikte dikkat edicek olursanız Kuvvetle Türlendirilmiş Veri Kümemize ait metodlar ve nesneler kullanılmıştır. Örneğin, bir DataTable nesnesi ile referans edilen bellek bölgesindeki tabloya, yeni bir satır eklemek için öncelikle yeni bir DataRow nesnesi tanımlarız. Sonra bu DataRow nesnesini boş bir satır olacak şekilde DataTable nesnemiz için oluştururuz. Daha sonra bu DataRow nesnesinde her bir alan için yeni verileri ekleriz. Son olarakta bu oluşturulan yeni DataRow nesnesini DataTable'ımıza ekleriz.

Böylece, DataTable nesneminin işaret ettiği tabloya yeni bir satır eklemiş oluruz. Bu teknik klasik olarak Türleştirilmemiş Veri Kümelerini kullandığımız örneklerde aşağıdaki kodlar ile gerçekleştirilmektedir.

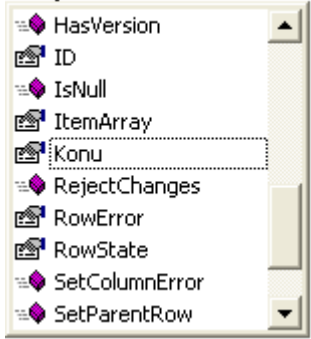
```
DataRow drKlasik;
drKlasik=ds.Tables[0].NewRow();
drKlasik[1]=txtKonu.Text;
drKlasik[2]=Convert.ToDateTime(txtTarih.Text);
drKlasik[3]=txtAdres.Text;
ds.Tables[0].Rows.Add(drKlasik);
```

Gördüğümüz gibi teknik olarak iki yaklaşımda aynıdır. Ancak, aralarındaki farkı anlamak için kullanılan ifadelerle yakından bakmamız yeterlidir. Herşeyden önce Kuvvetle Türleştirilmiş Veri Kümelerini kullanmak, kod yazarken programcıya daha anlaşılır gelmektedir. Dilerseniz, bu iki tekniği aşağıdaki tablo ile karşılaştıralım.

UnTyped Dataset Tekniği	Typed DataSet Tekniği
Yeni Bir Satır Tanımlamak	
DataRow drKlasik;	<pre>dsMakale.MakaleRow dr;</pre>  Görüldüğü gibi intelliSense özelliği sayesinde, dsMakale dataSet'inden sonra yeni bir DataRow nesnesi oluşturmak için gerekli söz dizimini bulmak ve anlamak son derece kolay.
Tanımlanan Yeni Satırı Oluşturmak	
drKlasik=ds.Tables[0].NewRow();	dr=mk.Makale.NewMakaleRow();

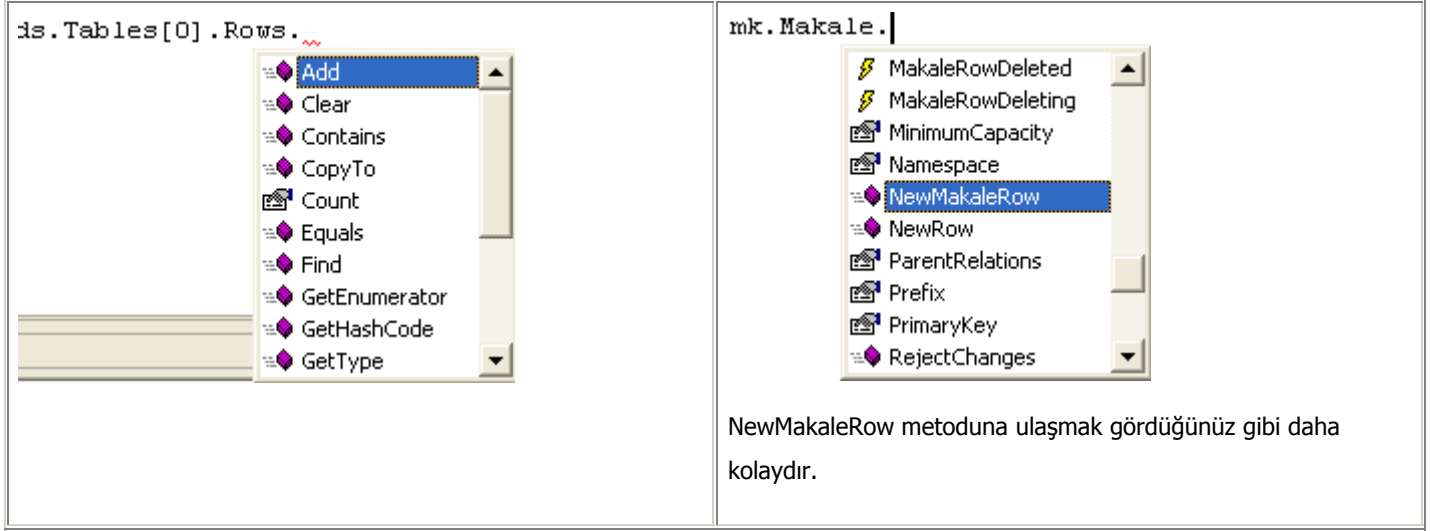
	 Görüldüğü gibi, Typed DataSet sınıfımız yardımıyla tanımlanan yeni satırı oluşturmak için kullanacağımız söz dizimi çok daha okunaklı ve anlamlı.

Alanlara Verileri Aktarmak

<pre>drKlasik[1]=txtKonu.Text; drKlasik[2]=Convert.ToDateTime(txtTarih.Text); drKlasik[3]=txtAdres.Text; drKlasik[▲ 6 of 6 ▼ object DataRow [int columnIndex]</pre>	<pre>dr.Konu=txtKonu.Text; dr.Tarih=Convert.ToDateTime(txtTarih.Text); dr.Adres=txtAdres.Text;</pre>  Bir Type DataSet üzerinden oluşturduğumuz DataRow nesnesinin alanlarına veri aktarırken hangi alanların olduğunu kolayca gözlemleyebiliriz. Üstelik bu biçim, kodumuza daha kolay okunurluk ve anlam kazandırır.

Oluşturulan Satırın Tabloya Eklenmesi

<pre>ds.Tables[0].Rows.Add(drKlasik);</pre>	<pre>mk.Makale.AddMakaleRow(dr);</pre>



Tablo 1. Türleştirilmiş (Typed) ve Türleştirilmemiş (UnTyped) Veri Kümelerinde Satır Ekleme İşlemlerinin Karşılaştırılması.

Her iki teknik arasında kavramsal olarak fark bulunmamasına rağmen, uygulanabilirlik ve kolaylık açısından farklar olduğunu görüyoruz. Bu durum verileri ararken, düzenlerken veya silerkende karşımıza çıkmaktadır. Şimdi uygulamamızda belli bir satırı nasıl bulacağımızı inceleyelim. Kullanıcı Bul başlıklı butona tıkladığında txtMakaleID textBox kontrolüne girdiği değerdeki satır bulunacak ve bulunan satıra ait alan verileri, formumuzdaki diğer textBox kontrollerine yüklenecek. Arama işleminin PrimaryKey üzerinden yapıldığını belirtelim. Şimdi kodlarımızı yazalım.

```
private void btnBul_Click(object sender, System.EventArgs e)
{
    dsMakale.MakaleRow drBulunan; /* Arama sonucu bulunan satırı tutacak DataRow nesnemiz tanımlanıyor. */
    drBulunan=mk.Makale.FindByID(Convert.ToInt32(txtMakaleID.Text)); /* FindByID metodu, Türleştirilmiş Veri Kümemizdeki
    tablomuzun Primary Key alanı üzerinden arama yapıyor ve sonucu drBulunan DataRow(MakaleRow) nesnemize atıyor. */
    if(drBulunan!=null) /* Eğer aranan satır bulunursa drBulunan değeri null olmayacaktır. */
    {
        txtKonu.Text=drBulunan.Konu; /* Bulunan satıra ait alan verileri ilgili kontrollere atanıyor. */
        txtTarih.Text=drBulunan.Tarih.ToString();
        txtAdres.Text=drBulunan.Adres;
    }
    else
    {
        MessageBox.Show("Aranan Makale Bulunamadı");
    }
}
```

Bu arama tekniğinin, türleştirilmemiş veri kümelerindeki arama tekniğine göre çok farklı bir özelliği vardır. Klasik yöntemde bir DataTable üzerinden arama yaparken Find metodunu kullanırsınız.

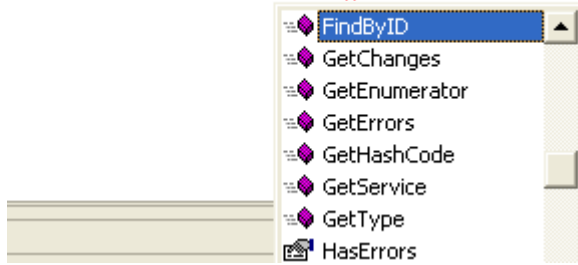
```
dr=dt.Rows.Find(|
```

2 of 2 System.Data.DataRow DataRowCollection.Find (object key)
key: The primary key value of the System.Data.DataRow to find.

Şekil 3. Klasik Find metodu.

Görüldüğü gibi Find metodu, PrimaryKey değerini alır ve bu değeri kullanarak, tablonun PrimaryKey alanı üzerinden arama yapar. Oysa örneğimizde kullandığımız Türleştirilmiş Veri Kümesine ait Makale DataTable nesnesinin Find metodu isim değişikliğine uğrayarak FindByID haline dönüşmüştür. Nitekim, bu yeni DataTable nesnesi hangi alanın PrimaryKey olduğunu bilir ve metodun ismini buna göre değiştirerek arama kodunu yazmamızı son derece kolaylaştırır.

```
dsMakale.MakaleRow drBulunan;  
drBulunan=mk.Makale.F
```



Şekil 4. FindByID metodu;

Diğer yandan, Klasik Find metodumuz, key parametresini object türünden alırken, FindByID metodumuz PrimaryKey alanının değeri ne ise, parametreyi o tipten alır. Buda FindByID metodunun, klasik Find metodundanki object türünden dolayı, daha performanslı çalışmasına neden olur.

```
drBulunan=mk.Makale.FindByID(  
TypeDataSet3.dsMakale.MakaleRow MakaleDataTable.FindByID (int ID)
```

Şekil 5. FindByID metodunda parametre tipi.

Diğer yandan aynı işi yapan bu metodlar arasındaki fark birden fazla Primary Key alanına sahip olan tablolarda daha çok belirginleşir. Söz gelimi, Sql sunucusunda yer alan Northwind veri tabanındaki, Order Details tablosunun hem OrderID alanı hemde ProductID alanı Primary Key'dir. Dolayısıyla bu iki alan üzerinden arama yapmak istediğimizde klasik Find metodunu aşağıdaki haliyle kullanırız.

```
DataRow drBulunan;  
drBulunan = dt.Find(new object[] {10756,9});
```

Oysaki bu tabloyu Türleştirilmiş Veri Kümesi üzerinden kullanırsak kod satırları aşağıdakine dönüşür.

```
drBulunan=ds.SiparisDetay.FindByOrderIDProductID(10756,9);
```

Gelelim verilerin düzenlenmesi işlemine. Şimdi uygulamamızda bulduğumuz satıra ait verileri textBox kontrollerine aktarıldıktan sonra, üzerlerinde değişiklik yaptığımızda bu değişikliklerin DataTable'ımıza nasıl yansıtacağımıza bakalım. Normal şartlarda, Türleştirilmemiş Veri Kümesi üzerindeki bir DataTable'a ait herhangi bir satırda yapılan değişiklikler için, güncel DataRow nesnesine ait BeginEdit ve EndEdit metodları kullanılmaktadır. Oysaki Türleştirilmiş Veri Kümelerindenki satırlara ait alanlar birer özellik olarak

tutulduklarından, sadece bu özelliklere yeni değerlerini atamamız yeterli olmaktadır. Bu veri kümesinin bir sınıf şeklinde tutuluyor olmasının sağlamış olduğu güçtür. Bu nedenle Türleştirilmiş Veri Kümeleri Strongly takısını haketmektedir. Dilerseniz uygulamamızda arama sonucu elde ettiğimiz bir satıra ait alan değerlerini güncelleyeceğimiz kod satırlarını yazalım.

```
private void btnDegistir_Click(object sender, System.EventArgs e)
{
    drBulunan.Konu=txtKonu.Text;
    drBulunan.Adres=txtAdres.Text;
    drBulunan.Tarih=Convert.ToDateTime(txtTarih.Text);
}
```

Şimdi uygulamamızı çalıştıralım. ID değeri 6 olan satırı bulalım. Daha sonra bu satırdaki bazı veileri değiştirelim ve Degistir başlıklı butona tıklayalım. Değişikliklerin hemen gerçekleştiğini ve DataGrid kontrolünde yansıdığını görürüz.

ID	Konu	Tarih	Adres
3	Web Sayfalarında Stored	12.11.2003	makale\inwebpagestoredpro
4	Overload Metodların Güç	13.11.2003	makale\overloading.htm
5	Transaction Kavramı	17.11.2003	makale\transaction.htm
6	Distributed Transactions	19.11.2003	makale\distransaction.htm
7	Bir Form ve Kontrollerinin	21.11.2003	makale\manuelCreateForm.

Konu: Distributed Transactions

Tarih: 19.11.2003 00:00:00

Adres: makale\distransaction.htm

Ekle

ID: 6

Bul

Degistir

Şekil 6. ID=6 olan satır bulunuyor.

ID	Konu	Tarih	Adres
2	Stored Procedure Ya	12.11.2003	makale\storedprocdelete.ht
3	Web Sayfalarında St	12.11.2003	makale\inwebpagestoredpro
4	Overload Metodların	13.11.2003	makale\overloading.htm
5	Transaction Kavram	17.11.2003	makale\transaction.htm
6	Distributed	19.11.2010	makale\distran.htm

Konu:

Tarih:

Adres:

ID:

Şekil 7. Güncel Satır Verileri Değiştiriliyor.

Son olarak satır silme işlemini inceleyelim. Bu amaçla RemoveMakaleRow metodunu kullanacağız. Elbette bu metod, örneğimizde oluşturduğumuz Kuvvetle Türlendirilmiş Veri Kümemizin sınıfı içinde yeniden yazılmış bir metoddur ve sınıfımız içindeki prototipide şu şekildedir.

```
public void RemoveMakaleRow(MakaleRow row)
{
    this.Rows.Remove(row);
}
```

Açıkçası yaptığı işlem sınıfın Rows koleksiyonundan row parametresi ile gelen satırı çıkartmaktır. Uygulamamızda ise bu metodu aşağıdaki şekilde kullanırız.

```
private void btnSil_Click(object sender, System.EventArgs e)
{
    mk.Makale.RemoveMakaleRow(drBulunan);
}
```

Bu metod parametre olarak aldığı satırı tablodan çıkartır.

Böylece Kuvvetle Türlendirilmiş Veri Kümeleri üzerinden satır ekleme, arama, düzenleme ve silme işlemlerinin nasıl yapılacağını incelemiş olduk. Umuyorumki hepiniz için faydalı bir makale olmuştur. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

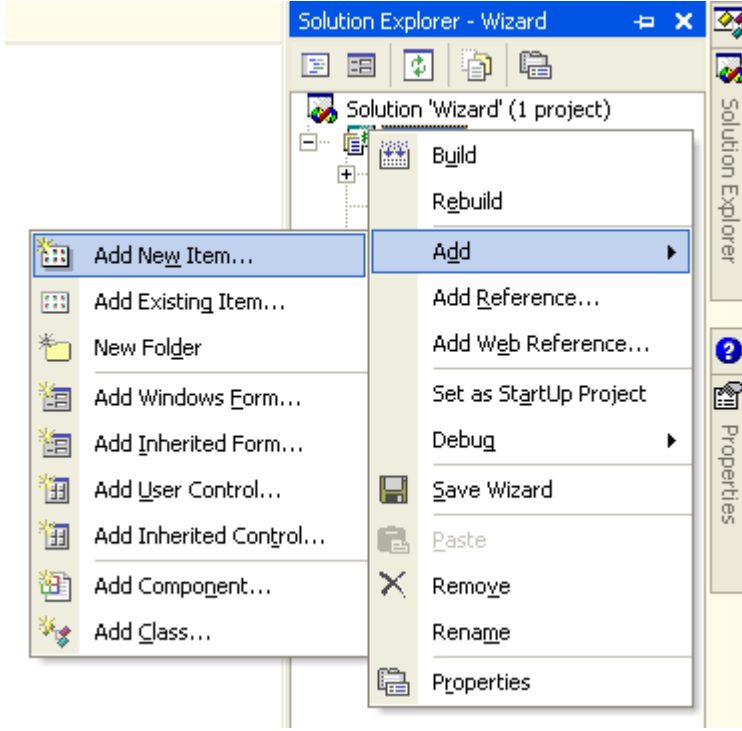
Data Form Wizard Yardımıyla İlişkili Tablo Uygulamalarının Hazırlanması

Değerli Okurlarım, Merhabalar.

Bu makalemizde, Visual Studio.NET ortamında, Data Form Wizard yardımıyla, veritabanı uygulamalarının ne kadar kolay bir şekilde oluşturulabileceğini inceleyeceğiz. Pek çok programcı, uygulamalarını geliştirirken sihirbazları kullanmaktan açıkça kaçınır. Bunun bir nedeni, sihirbazların işlemleri çok fazla kolaylaştırması ve programcıyı tembelliğe itmesidir. Gerçektende, bir Data Form Wizard yardımıyla uzun sürede programlayacağınız bir veritabanı uygulamasını inanılmaz kısa sürede tamamlayabilirsiniz. Diğer yandan, bir programcı için bir uygulamayı geliştirmekteki en önemli unsur belkide şu kelimenin arkasında gizlidir; Kontrol.

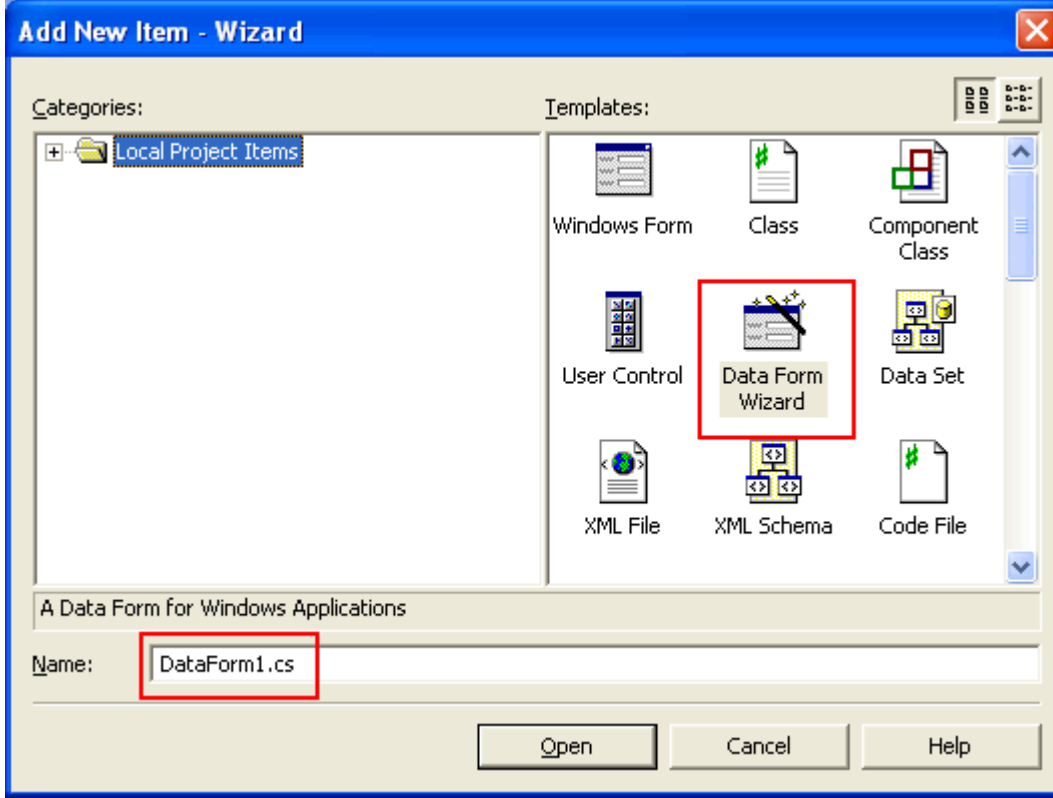
Kontrol bir programcı için, uygulamanın her yerinde hakim olmak demektir. Yazılabilecek her kodun programcı tarafından yazılması, olabilecek tüm hataların düzeltilmesi, mantıksal bütünlüklerin sağlanması ve kullanıcının ihtiyaçlarına en üst düzeyde cevap verilebilmesi, programcının kontrolünü güçlendiren unsurlar arasında yer alır. Gerçektende ben size, gerçek hayatta sihirbazları çok fazla kullanmamanızı tavsiye ederim. Herşeyden önce, sihirbazlar tek düzelik sağlarlar ve sürekli aynı adımları atarlar. Bu bir süre sonra hem sizi tembelleştirecek hemde gerçek bir programcı gibi düşünmekten örneğin oluşabilecek programatik hataların önceden sezilebilmesi yetenğinden mahrum bırakacaktır.

Ancak tüm bu olumsuzlukların yanında, bir Data Form Wizard aracını kullanarak, .NET ortamında veritabanı programlamasını öğrenmeye çalışan bir programcı için, neler yapılabileceği? , bunların hangi temel adımlarla gerçekleştirildiği? , hatta Visual Studio.NET tarafından otomatik olarak oluşturulan kodların nerelerde devreye girdiği? gibi sorunların anlaşılmasında oldukça kolay olacaktır. Diğer yandan profesyonel programcılarda zaman zaman, sadece kendileri için bir takım verileri kolayca izleyebilmek amacıyla yada ani müdahalelerde bulunmak amacıyla oluşturacakları uygulamalarda aynı kodları tekrardan yazmak yerine sihirbazları kullanmayı tercih edebilirler. Dolayısıyla zaman zaman sihirbazları kullanmak oldukça işe yarayabilir ama bunu alışkanlık halinede getirmemek gerekir. İşte bu makalemizde, bir Ado.Net uygulamasının, ilişkili iki tablo için nasıl kolayca oluşturulabileceğini inceleyeceğiz. Uygulamamız sona erdiğinde, tabloları izleyebilecek, ilişkili kayıtları inceleyebilecek, yeni kayıtlar ekleyebilecek, var olanları düzenleyip silebilecek ve kayıtlar arasında gezinebileceğiz. Ayrıca sihirbazımız, programatik olarak oluşturulan Ado.Net uygulamalarında da hangi adımları takip edeceğimiz hakkında bize kılavuzluk etmiş olacak. Şimdi dilerseniz uygulamamızı yazmaya başlayalım. Öncelikle Visual Studio ortamında bir C# Windows Uygulaması açalım. Daha sonra veritabanı uygulamamızı taşıyacak formu eklemek için, Solution sekmesinde, projemize sağ tuş ile tıklayalım ve Add New Item öğesini seçelim.



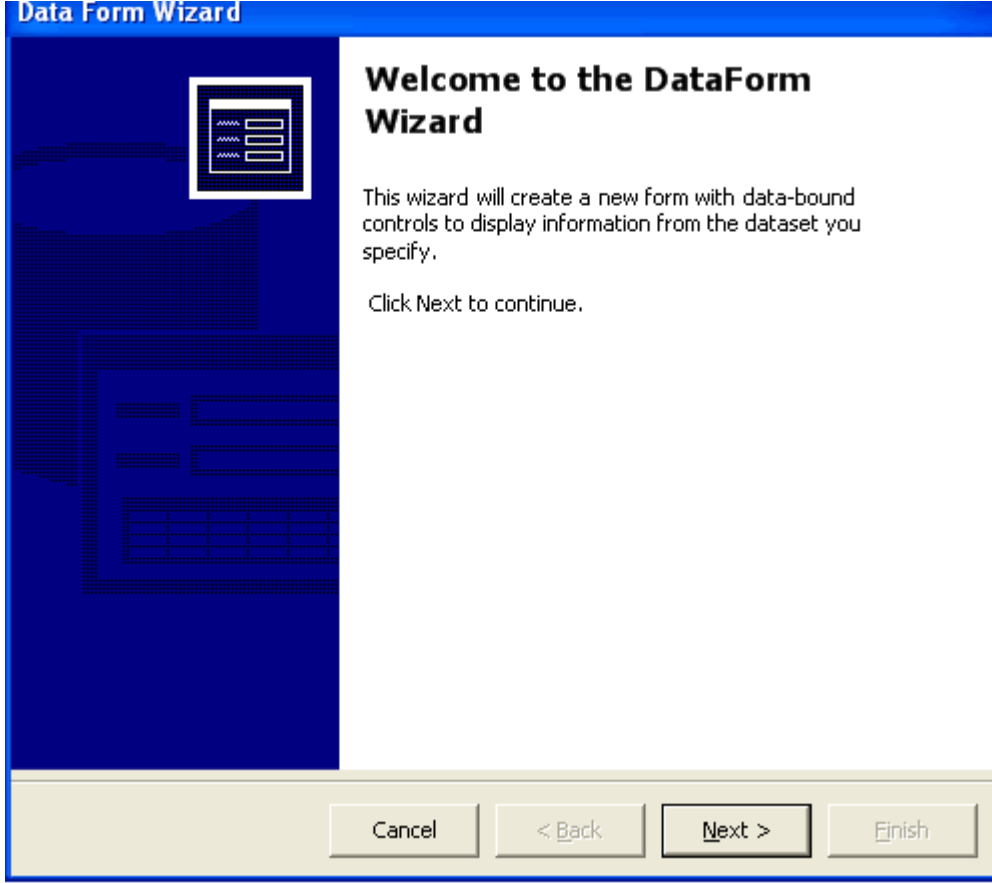
Şekil 1. Add New Item.

Bu durumda karşımıza aşağıdaki pencere gelecektir. Bu pencerede Data Form Wizard aracını seçelim. Data Form Wizard aracı tüm adımları bitirildikten sonra, projemize cs uzantılı bir sınıf ve bu sınıfı kullanan bir Form ekleyecektir. Veritabanına bağlanması, tabloların doldurulması, satırlar arasında gezinme gibi pek çok işlevi yerine getiren metodlar, özellikler ve nesneler, Data Form Wizard ile oluşturulan bu sınıf içerisine yazılacaktır. Burada Form dosyanıza anlamlı bir isim vermenizi öneririm.



Şekil 2. Data Form Wizard

Bundan sonra, artık uygulamayı oluşturacağımız adımlara geçmiş oluruz.



Şekil 3. Adımlara başlıyoruz.

İlk adımımız DataSet nesneminin isimlendirilmesi. Bir DataSet, veritabanından bağımsız olarak uygulamanın çalıştığı sistemin belleğinde oluşturulan bir alana referans eden kuvvetli bir nesnedir. Bir DataSet içersine, tablolar, tablolar arası ilişkiler vb. ekleyebiliriz. Bir DataSet ile, veritabanındanki gerekli veriler DataSet nesnesinin bellekte temsil ettiği bölgeye yüklendikten sonra bu veritabanına bağlı olmaksızın çalışabiliriz. Dahada önemlisi, veritabanına bağlı olmaksızın, veriler ekleyebilir, verileri düzenleyebilir, silebilir, yeni tablolara, görünüm, ilişkiler oluşturabiliriz. DataSet, ADO.NET 'in en önemli yeniliklerinden birisi olmakla birlikte, bağlantısız katman dediğimiz kısmın bir parçasıdır. DataSet'i ilerleyen makalelerimizde daha detaylı olarak da inceleyeceğiz. Şu an için bilmeniz gereken, oluşturacağımız DataSet'in Sql sunucumuzda (veya başka bir databasede) yer alan ilişkili tablolarımızı ve aralarındaki ilişkiyi içerecek olan bağlantısız (ADO' daki RecordSet gibi, Veritabanına sürekli bir bağlantıyı gerektirmeyen) bir nesne oluşudur. Şimdi burada uygulamaya önceden eklediğimiz bir DataSet nesnesini seçebileceğimiz gibi yeni bir tanede oluşturabiliriz. Biz yeni bir DataSet oluşturacağımızdan, Creat a new database named alanına, DataSet'imizin adını giriyoruz.

Data Form Wizard

Choose the dataset you want to use
The dataset contains the tables and columns to work with in your form.

The wizard helps you create a new dataset. Alternatively, you can use a dataset already defined in your project or one provided by a Web service.

Which dataset do you want to use?

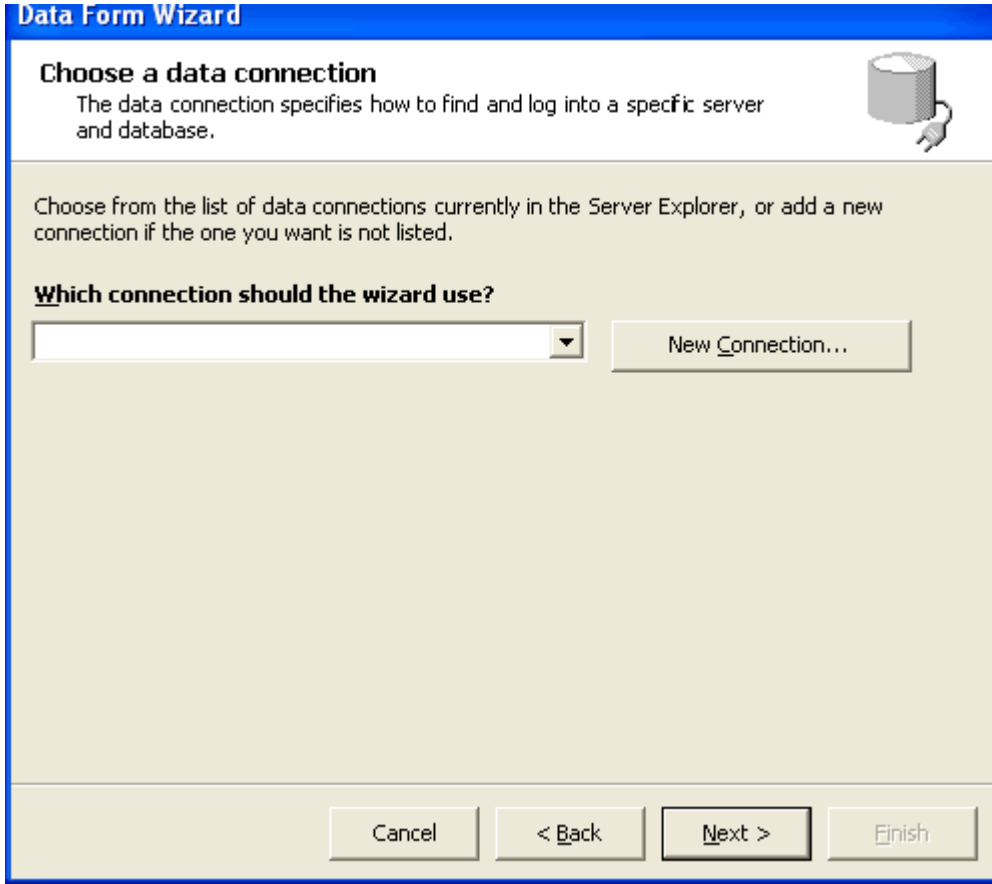
☒ Create a new dataset named:
dsSatislar

☐ Use the following dataset:
[Dropdown Menu]

Cancel < Back Next > Finish

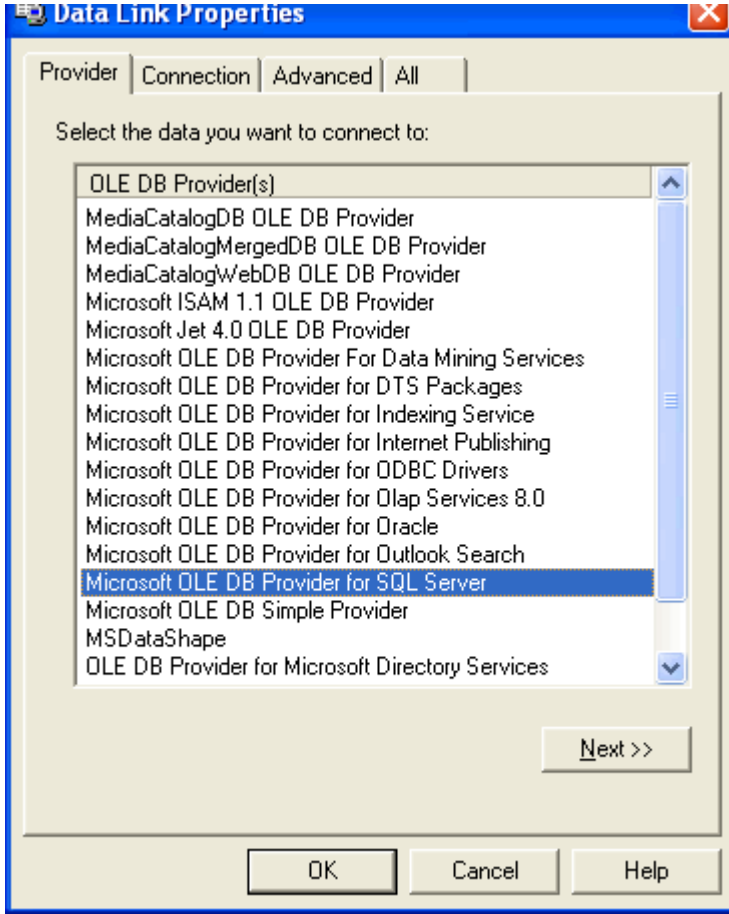
Şekil 4. İlk adımımız DataSet nesnemizi belirleyeceğimiz ad ile oluşturmak.

Sıradaki adım, veritabanına olan bağlantımızı gerçekleştirecek nesnemizi tanımlayacak. Herşeyden önce DataSet nesnemize bir veritabanından veriler yükleyeceksek, bir veri sağlayıcısı üzerinden bu verileri çekmemiz gerekecektir. İşte bu amaçla, pek çok ADO.NET nesnesi için gerekli olan ve belirtilen veri sağlayıcısı üzerinden ilgili veri kaynağına bir bağlantı açmamıza yarayan bir bağlantı (Connection) nesnesi kullanacağız. Burada uygulamamızda Sql Sunucumuzda yer alan veritabanına OleDb üzerinden erişmek istediğimiz için, sihirbazımız bize bir OleDbConnection nesnesi örneği oluşturacaktır. Veri sağlayıcısının tipine göre, OleDbConnection, SqlConnection, OracleConnection, OdbcConnection bağlantı nesnelerini oluşturabiliriz. New Connection seçeneğini seçerek yeni bir bağlantı oluşturmamızı sağlayacak diğer bir sihirbaza geçiş yapıyoruz.



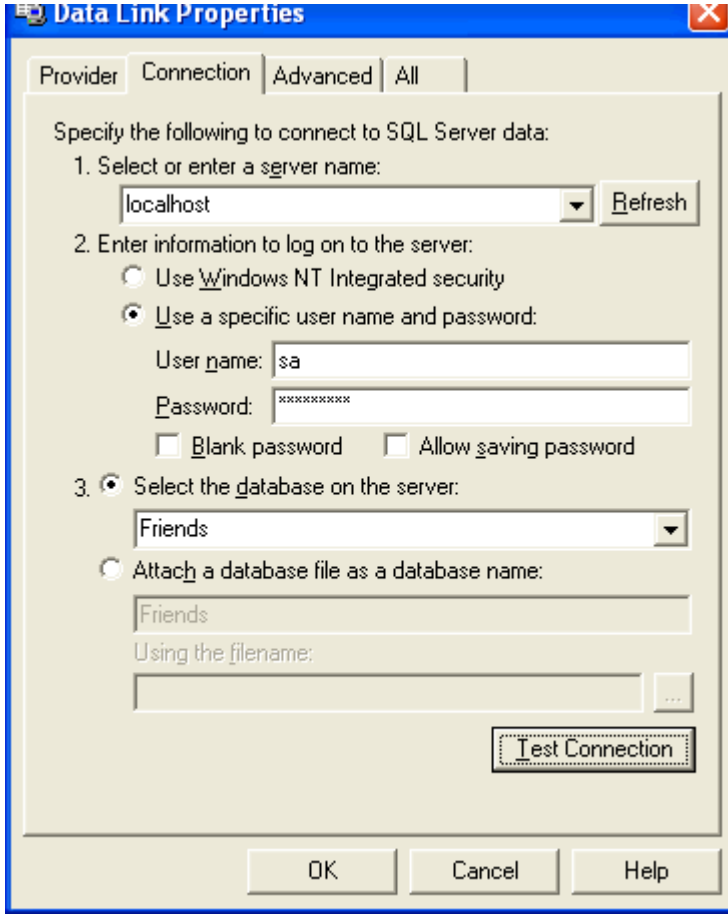
Şekil 5. Connection nesnemizi oluşturmaya başlıyoruz.

Şimdi karşımızda Data Link Properties penceresi yer almakta. Burada, Provider kısmında Microsoft OLE DB Provider For Sql Server seçeneği seçili olmalıdır.



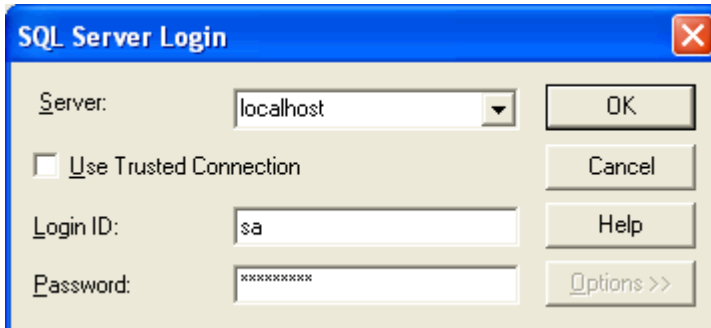
Şekil 6. Provider.

Connection sekmesinde ise öncelikle bağlanmak istediğimiz sql sunucusunun adını girmeliyiz. Sql sunucumuz bu uygulamada, istemci bilgisayar üzerinde kurulu olduğundan buraya localhost yazabiliriz. 2nci seçenekte, sql sunucusuna log in olmamız için gerekli bilgileri veriyoruz. Eğer sql sunucumuz windows authentication'ı kullanıyorsa, Use Windows Integrated Security seçeneğini seçebiliriz. Bu durumda sql sunucusuna bağlanmak için windows kullanıcı bilgileri kontrol edilecektir. Bunun sonucu olarak username ve password kutucukları geçersiz kılınacaktır. Diğer yandan sql sunucusuna, burada oluşturulmuş bir sql kullanıcı hesabı ile de erişebiliriz. Ben uygulamamda sa kullanıcıyı kullandım. Bildiğiniz gibi sa sql kullanıcısı admin yetkilerine sahip bir kullanıcı olarak sql sunucusunun kurulması ile birlikte sisteme default olarak yüklenir. Ancak ilk yüklemede herhangi bir şifresi yoktur. Güvenlik amacıyla bu kullanıcıya mutlaka bir şifre verin.(Bunu sql ortamında gerçekleştireceksiniz.) Allow Saving Password seçeneğini işaretlediğimiz takdirde, uygulamayı kim kullanıyor ise, uygulama başlatıldığında sql sunucusuna bağlanabilmek için kendisinden bu kullanıcı adına istinaden şifre sorulacaktır. Eğer bu seçeneği işaretlersek, program çalıştırıldığında buraya girilen şifre sorulmayacaktır. Şimdi 3nci seçeneğe geçersek ve combo box kontrolünü açarsak sql sunucusunda yer alan veritabanlarının tümünü görebiliriz. Buradan kullanacağımız veritabanını seçeceğiz.



Şekil 7. Connection bilgileri giriliyor.

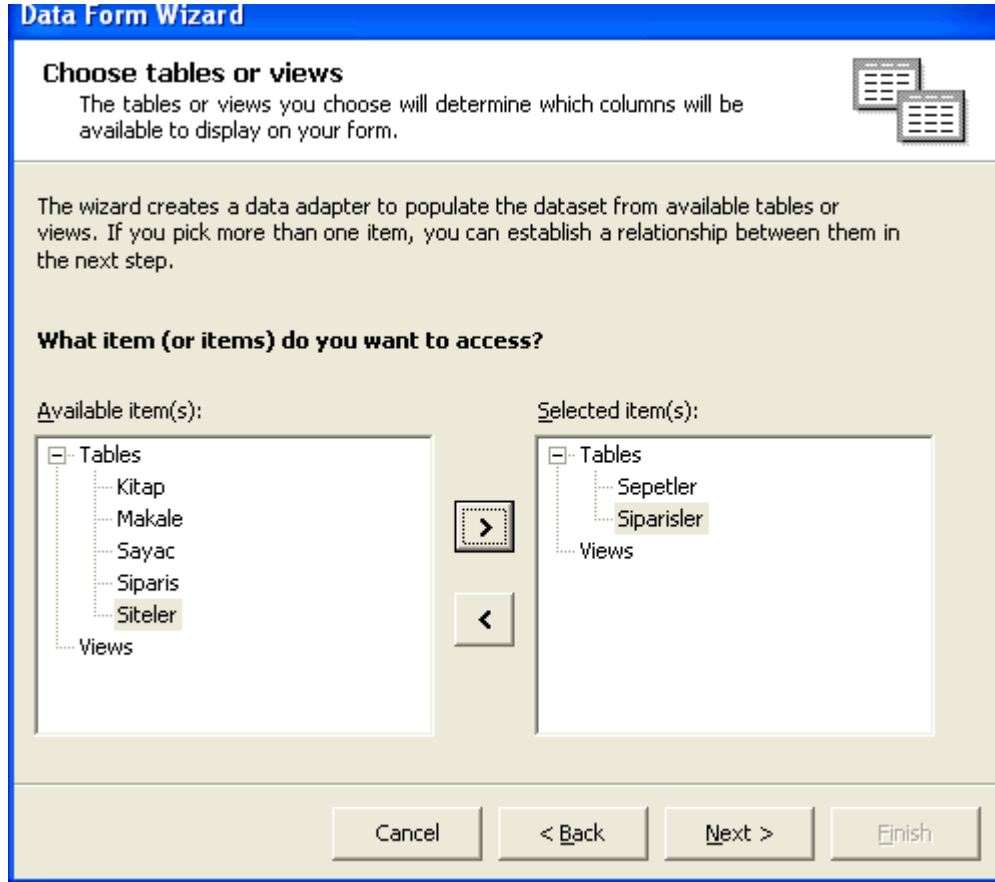
Dilersek Test Connection butonuna tıklayarak, sql sunucusuna bağlanıp bağlanamadığımızı kontrolde edebiliriz. Şimdi OK diyelim ve Data Link Properties penceresini kapatalım. Bu durumda karşımıza bir Login penceresi çıkacaktır. Bu az önce Allow Saving Password seçeneğini işaretlememiş için karşımıza gelmiştir. Şifremizi tekrardan girerek bu adımı geçiyoruz. Bu durumda, sql sunucumuz için gerekli bağlantıyı sağlayacak connection nesnemizde oluşturulmuş olur.



Şekil 8. Login penceresi.

Sıradaki adımımızda ise, uygulamamızda kullanacağımız tabloları seçeceğiz. Bu adımda, bağlandığımız veritabanında yer alan tablolar ve görünümüler yer almaktadır. Biz hemen ekleyeceğimiz iki tabloyu seçip pencerenin sağ tarafında yer alan Selected

Item(s) kısmına aktarıyoruz. Bu tabloların her biri birer DataTable nesnesi olacak şekilde DataSet nesnemize eklenir. DataTable nesneleri tablolara ait bilgilerinin ve verilerinin bellekte yüklendiği belli bir alanı referans ederler. Bir DataSet nesnesi birden fazla DataTable içerebilir. Böylece veritabanı ortamını DataSet nesnesi ile uygulamanın çalıştığı sistem belleğinde simüle etmiş oluruz.



Şekil 9. Tablolar seçiliyor.

Bu iki tablomuz hakkında yeri gelmişken kısaca bilgi verelim. Sepetler tablosunda alışveriş sepetlerine ait bilgiler yer alıyor. Siparisler tablosu ise bu belirli sepetin içinde yer alan ürünlerin listesini tutuyor. Yani Sepetler tablosundan Siparisler tablosuna doğru bireçok ilişki söz konusu. Bu adımda tamamladıktan sonra, veri kaynağına log in olmamız için tekrar bir şifre ekranı ile karşılaşacağız. Burayıda geçtikten sonra sıra, iki tablo arasındaki ilişkiyi kuracağımız adıma geldi. Burada aralarında ilişki oluşturmak istediğimiz tabloları ve ilgili kolonları seçerek ilişkimiz için bir isim belirleyeceğiz. Bunun sonucu olarak sihirbazımız bir DataRelation nesnesi oluşturacak ve bu nesneyi DataSet'e ekleyecek. Böylece veritabanından bağımsız olarak çalışırken, DataSet kümemiz bu DataRelation nesnesini kullanarak tablolar arasındaki bütünlüğünde sağlamış olacak. Öncelikle bu ilişki için anlmalı bir ad belirleyin. Çoğunlukla ilişkinin hangi tablodan hangi tabloya doğru olduğuna bağlı olarak bir isimlendirme yapılır. Daha sonra Parent(Master) tablo ve Primary Key seçilir. Ardından Foreign Key alanını içeren tabloda, Child(Detail) tablo olarak seçilir. Uygulamamızda Sepetler tablosu ile Siparisler tablosu SepetID alanları ile birbirlerine ilişkilendirilmiştir.

Data Form Wizard

Create a relationship between tables
The wizard will use the relationships to generate code that keeps the tables synchronized as you work with them.

Relationships are based on common keys between tables. Name your new relation, choose the parent and child tables and key fields, and then add it to the relations list using the arrow button.

Name:

Parent table: Child table:

Keys:

Relations:

Şekil 10. DataRelation nesnesi oluşturuluyor.

Bir sonraki adımımızda tabloların hangi alanlarının Form üzerinde görüntüleneceğini belirleriz. Başlangıç için bütün alanlar seçili haldedir. Bizde bunu uygulamamızda bu halde bırakacağız.

Data Form Wizard

Choose tables and columns to display on the form
Your form can display any of the tables and columns available in the dataset.

If you display more than one table on the form, the tables have a master-detail relationship

What tables and columns do you want to display on the form?

Master or single table: Detail table:

Columns:

☒	SepetID
☒	SepetSahibi
☒	SepetTarihi
☒	SepetTutari

Columns:

☒	UrunID
☒	SepetID
☒	Miktar
☒	BirimFiyat

Cancel < Back Next > Finish

Şekil 11. Görüntülenecek Alanların Seçilmesi.

Son adım ise, Formun tasarımının nasıl olacağıdır. İstersek master tabloyu DataGrid kontrolü içinde gösterebiliriz. Ya da Master tabloya ait verileri bağlanmış kontrollerle gösterebiliriz. Bunun için Single record in individual controls seçeneğini seçelim. Diğer seçenekleride olduğu gibi bırakalım.

Data Form Wizard

Choose the display style
You can display a single record at a time or display all records at once.

How do you want to display your data?

☐ All records in a grid

☒ Single record in individual controls

What additional controls do you want on the form?

☒ **Cancel All** - Cancels changes to all records in the dataset.

If you select individual controls, you can add controls for editing and navigation:

☒ **Add** - Creates a new record.

☒ **Delete** - Deletes the current record.

☒ **Cancel** - Cancels changes to the current record.

☒ **Navigation controls** - Moves to first, previous, next, or last record.

The wizard now has the information it needs. Click Finish to exit and generate your new form.

Cancel < Back Next > Finish

Şekil 12. Son Adım. Formun görüntüsünün ayarlanması.

Böylece Data Form Wizard yardımıyla veritabanı uygulamamız için gerekli formu oluşturmuş olduk. Karşımıza aşağıdaki gibi bir Form çıkacaktır.

Şekil 13. Formun son hali.

Görüldüğü gibi tüm kontrolleri ile birlikte uygulamamız oluşturulmuştur. Sql sunucumuza Ole Db veri sağlayıcısı üzerinden erişmemizi sağlayan OleDbConnection nesnemiz, tablolardaki verileri DataSet nesnesine yüklemek (Fill) ve bu tablolardaki değişiklikleri, veritabanına yansıtmak (update) için kullanılan OleDbDataAdapter nesneleri, tabloları ve tablolar arası ilişkiyi bellekte sakayıp veritabanından bağımsız olarak çalışmamızı sağlayan DataSet nesnemiz ve diğerleri... Hepsi sihirbazımız sayesinde kolayca oluşturulmuştur. Gelelim bu form ile neler yapabileceğimize.

Her şeyden önce uygulamamızı çalıştırdığımızda hiç bir kontrolde verilerin görünmediğine şahit olacaksınız. Önce, veritabanından ilgili tablolara ait verilerin DataSet üzerinden DataTable nesnelere yüklenmesi gerekir. Bu Fill olarak tanımlanan bir işlemdir. Load butonu bu işlemi gerçekleştirir. Load başlıklı butonun koduna bakarsak;

```
private void btnLoad_Click(object sender, System.EventArgs e)
{
    try
    {
        // Attempt to load the dataset.
        this.LoadDataSet();
    }
}
```

```

catch (System.Exception eLoad)
{
    // Add your error handling code here.
    // Display error message, if any.
    System.Windows.Forms.MessageBox.Show(eLoad.Message);
}
this.objdsSatislar_PositionChanged();
}

```

Görüldüğü gibi kod LoadDataSet adlı bir procedure'e yönlendirilir.

```

public void LoadDataSet()
{
    // Create a new dataset to hold the records returned from the call to FillDataSet.
    // A temporary dataset is used because filling the existing dataset would
    // require the databindings to be rebound.
    Wizard.dsSatislar objDataSetTemp;
    objDataSetTemp = new Wizard.dsSatislar();
    try
    {
        // Attempt to fill the temporary dataset.
        this.FillDataSet(objDataSetTemp);
    }
    catch (System.Exception eFillDataSet)
    {
        // Add your error handling code here.
        throw eFillDataSet;
    }
    try
    {
        grdSiparisler.DataSource = null;
        // Empty the old records from the dataset.
        objdsSatislar.Clear();
        // Merge the records into the main dataset.
        objdsSatislar.Merge(objDataSetTemp);
        grdSiparisler.SetDataBinding(objdsSatislar, "Sepetler.drSepetlerToSiparisler");
    }
    catch (System.Exception eLoadMerge)
    {
        // Add your error handling code here.
    }
}

```

```

        throw eLoadMerge;
    }
}

```

Bu metotta da yükleme işlemi için bir DataSet nesnesi oluşturulur ve FillDataSet procedure'ü bu DataSet'i parametre olarak çağırılır. Bu kodlarda ise OleDbDataAdapter nesnelerinin Fill metodlarına tablolara ait verileri, veritabanından alarak DataTable nesnelerine yüklerler. Bunu yaparken OleDbDataAdapter nesneleri Select sorgularını kullanır. Bu sorgular, program içinde oluşturulmuş SqlCommand nesnelerinde saklanmaktadır. Her bir tablo için bir OleDbDataAdapter, bu tablodaki verileri almak için çalıştıracağı bir Sql sorgusunu, bir SqlCommand nesnesinden alır.

```

public void FillDataSet(Wizard.dsSatislar dataSet)
{
    // Turn off constraint checking before the dataset is filled.
    // This allows the adapters to fill the dataset without concern
    // for dependencies between the tables.
    dataSet.EnforceConstraints = false;
    try
    {
        // Open the connection.
        this.oleDbConnection1.Open();
        // Attempt to fill the dataset through the OleDbDataAdapter1.
        this.oleDbDataAdapter1.Fill(dataSet);
        this.oleDbDataAdapter2.Fill(dataSet);
    }
    catch (System.Exception fillException)
    {
        // Add your error handling code here.
        throw fillException;
    }
    finally
    {
        // Turn constraint checking back on.
        dataSet.EnforceConstraints = true;
        // Close the connection whether or not the exception was thrown.
        this.oleDbConnection1.Close();
    }
}

```

Uygulamanın diğer butonlarına ilişkin kodları da incelediğinizde her şeyin otomatik olarak sihirbaz tarafından uygulandığını göreceksiniz. Uygulamanın tüm kodlarını ekteki [DataForm1.cs](#) dosyası içinden inceleyebilirsiniz. Şimdi uygulamamızı çalıştıralım ve görelim. Uygulamayı çalıştırdığınızda Form1'in ekrana geldiğini ve DataForm1 in görünmediğini göreceksiniz. Bu durumu düzeltmek

için, projenin başlangıç formunu değiştirmemiz gerekiyor. Bunun için, Form1'in Main Procedure'ünün kodunu aşağıdaki gibi değiştirmemiz yeterlidir.

```
static void Main()
{
    Application.Run(new DataForm1());
}
```

Şimdi uygulamamızı çalıştırabiliriz. Load başlıklı button kontrolüne tıkladığımızda tablo verilerinin ekrana geldiğini ve kontrollere yüklendiğini göreceksiniz. Yön kontrolleri ile Sepetler tablosunda gezinirken DataGrid kontrolündeki verilerinde değiştiğine dikkatinizi çekmek isterim.

	UrunID	SepetID	Miktar	BirimFiyat
▶	4	1002	1	25000000
	5	1002	1	35000000
	6	1002	1	47500000
*				

Şekil 14. Uygulamanın çalışmasının sonucu.

Dilerseniz veriler üzerinde değişiklikler yapabilirsiniz. Ancak yaptığınız bu değişiklikler bellekteki DataSet kümesi üzerinde gerçekleşecektir. Bu değişiklikleri eğer, veritabanınada yazmak isterseniz Update başlıklı button kontrolüne tıklamanız gerekir. Bu durumda temel olarak, OleDbDataAdapter nesnesi, DataSet üzerindeki tüm değişiklikleri alacak ve veritabanını Update metodu ile

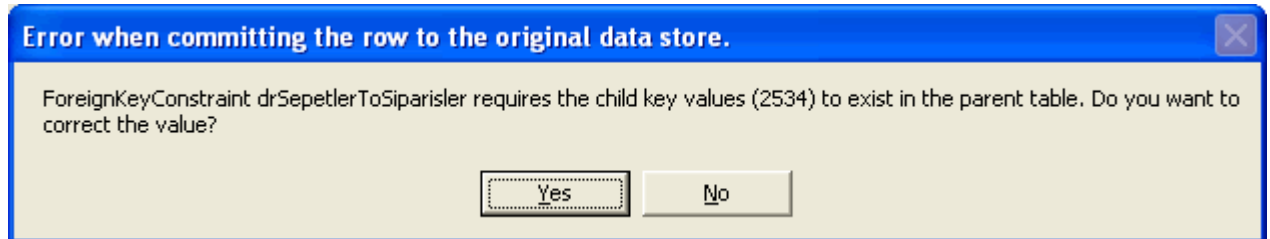
güncelleyecektir. Bu işlem için OleDbDataAdapter nesnesinin Update metodu, UpdateCommand özelliğine bakar. UpdateCommand özelliği, Update sorgusu içeren bir SqlCommand nesnesini işaret eder. Böylece güncelleme işlemi bu komut vasıtasıyla gerçekleşir. Bu işlemler yanında yeni satırlar ekleyebilir, var olanlarını silebilirsiniz. Gördüğü gibi komple bir veritabanı uygulaması oldu. Bunun dışında uygulamamızın güçlü yönleride vardır. Örneğin yeni bir Siparis girin. Ancak Siparis numarasını aşağıdaki gibi var olmayan bir SepetID olarak belirleyin.

Uyushmuyorlar

UrunID	SepetID	Miktar	BirimFiyat
1	1000	1	100000000
2	1000	4	12500000
9	2534	5	1501450

Şekil 15. Var olmayan bir ForeignKey girdik.

Bu durumda aşağıdaki uyarıyı alırsınız.



Şekil 16. ForeignKeyConstraint durumu.

Bu durumda Yes dersanız buradaki değer otomatik olarak 1000 yapılır. İşte bu olay DataRelation nesnemizin becerisidir. Değerli okurlarım uzun ama bir okadarda faydalı olduğuna inandığım bir makalemizin daha sonuna geldik. Hepinize mutlu günler dilerim.

GetOleDbSchemaTable Metodu İle Veritabanımızda Ne Var Ne Yok

Değerli Okurlarım, Merhabalar.

Bu makalemizde, OleDbConnection sınıfına atı olan GetOleDbSchemaTable metodu sayesinde, Sql Veritabanımızdaki varlıklara ait şema bilgilerini nasıl temin edebileceğimizi incelemeye çalışacağız. Çoğu zaman programlarımızda, bağlandığımız veritabanında yer alan tabloların (Tables), görünümünün (Views), saklı yordamların (Stored Procedures) ve daha pek çok veritabanı nesnesinin bir listesine sahip olmak isteriz. ADO.NET'te yer alan OleDbConnection nesnesine ait GetOleDbSchemaTable metodunu kullanarak bu istediğimiz sonuca varabiliriz. GetOleDbSchema metodu aşağıdaki prototipe sahiptir.

public DataTable **GetOleDbSchemaTable**(Guid *schema*,object[] *restrictions*);

Dikkat edecek olursanız metodun geri dönüş değeri DataTable tipindedir. Metodun isminde Table kullanılmasının nedenide zaten budur. Yani dönen şema bilgileri bir DataTable nesnesine aktarılacaktır. Metod iki önemli parametreye sahiptir. İlk parametremiz, OleDbSchemaGuid sınıfı türünden bir numaralandırıcı değeri almaktadır. Bu parametreye vereceğimiz değer ile, veritabanından elde etmek istediğimiz şema tipini belirleriz. Örneğin veritabanında yer alan tabloları elde etmek için, OleDbSchemaGuid.Tables değerini veririz. Bunun yanında bu parametreye verebileceğimiz başka önemli değerlerde şunlardır.

OleDbSchemaGuid Özelliği	Açıklaması
OleDbSchemaGuid .Columns	Tablo veya tablolara ait sütun yapısını sağlar.
OleDbSchemaGuid .Procedures	OleDbConnection nesnesinin bağlı olduğu veritabanında yer alan saklı yordamların listesini sağlar.
OleDbSchemaGuid .Views	OleDbConnection nesnesinin bağlı olduğu veritabanında yer alan görünümünün listesini sağlar.
OleDbSchemaGuid .Indexes	Belirtilen Catalog'da yer alan indexlerin listesini sağlar.
OleDbSchemaGuid .Primary_Keys	Belirtilen tablo veya tablolardaki birincil anahtarların listesini verir.

Tablo 1. OleDbSchemaGuid Üyelerinin Bir Kısmı

OleDbSchemaGuid sınıfı yukarıdaki örnek üyelerinin yanısıra pek çok üyeye sahiptir. Bu üyeler ile ilgili daha geniş bilgi için MSDN kaynaklarından faydalanmanızı tavsiye ederim. GetOleDbSchemaTable metodumuz ikinci bir parametre daha almaktadır. Bu parametre ile şema bilgisi üzerindeki sınırlamaları belirleriz. Bu sayede, ilk parametrede istediğimiz schema bilgilerinin sadece belirli bir tablo içinmi, yada veritabanının tamamı içinmi oluşturulacağına dair sonuçları elde etmiş oluruz. Örneklerimizi incelediğimizde bu parametrelerin ne işe yaradıklarını daha iyi anlayacağınızı düşünüyorum. Şimdi ilk örneğimizi geliştirelim. Basit bir Console uygulaması olarak geliştireceğimiz bu örnekte, Sql sunucumuzda yer alan, Friends isimli veritabanındaki tüm tabloların bir listesini elde edeceğiz. İşte program kodlarımız.

```
using System;
/* OleDb isim uzayını ve Data isim uzayını ekliyoruz. */
using System.Data;
using System.Data.OleDb;
namespace GetOleDbSchemaTables1
{
    class Class1
```

```

{
    static void Main(string[] args)
    {
        /* OleDbConnection nesnemizi oluşturuyoruz. SQLOLEDB provider'ını kullanarak, Sql sunucumuzda yer alan Friends
        isimli veritabanına bir bağlantı nesnesi tanımlıyoruz. */
        OleDbConnection con=new OleDbConnection("provider=SQLOLEDB;data source=localhost;initial
        catalog=Friends;Integrated Security=sspi");
        con.Open(); /* Bağlantımızı açıyoruz. */
        DataTable tblTabloListesi; /* GetOleDbSchemaTable metodunun sonucunu tutacak DataTable nesnemizi tanımlıyoruz.*/
        tblTabloListesi=con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,null); /* OleDbConnection nesnemizin,
        GetOleDbSchemaTable metodunu çalıştırıyoruz ve elde edilen şema bilgisini (ki bu örnekte Friends isimli veritabanındaki tüm
        tabloların listesini alıyor.) alıyoruz ve DataTable nesnemizin bellekte referans ettiği alana aktarıyoruz.*/
        foreach(DataRow dr in tblTabloListesi.Rows) /* DataTable'ımız içindeki satırlarda gezinebileceğimiz bir döngü
        oluşturuyoruz ve bu döngü içinde her bir satırın TABLE_NAME alanının değerini, dolayısıyla Friends veritabanındaki tablo adlarını
        ekrana yazdırıyoruz. */
        {
            Console.WriteLine(dr["TABLE_NAME"]);
        }
    }
}

```

Şimdi uygulamamızı çalıştırdığımızda Friends isimli veritabanında yer alan tüm tabloların ekrana yazıldığını görürsünüz.



Şekil 1. Friends veritabanındaki tabloların listesi.

Şimdi kodlarımızdaki `biTabloListesi=con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,null);` satırını daha yakından incelemeye çalışalım. İlk parametremiz `OleDbSchemaGuid.Tables`, `con` isimli `OleDbConnection` nesnemizin bağlandığı `Sql` sunucusundaki Friends veritabanından sadece tablo bilgilerini elde etmek istediğimizi göstermektedir. Sınırlandırıcı özelliğe sahip olan ikinci parametremize `null` değerini vererek tüm tabloların şemaya dahil edilmesini ve `DataTable` nesnesine aktarılmasını sağlamış olduk. Bu noktada bu iki parametrenin birbirleri ile ilişkil olduklarını söyleyebiliriz. Çünkü, `OleDbSchemaGuid` parametresinin vereceği tabloların şema yapısı, ikinci parametreye bağlıdır. Şöyleki;

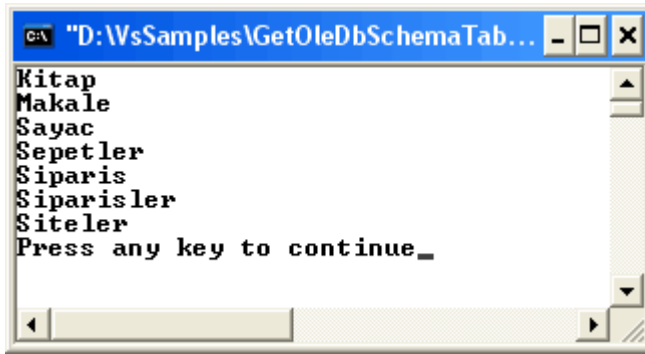
Sınırlama Alanı	Açıklaması
TABLE_CATALOG	Katalog adı. Eğer provider'ımız katalogları desteklemiyorsa <code>null</code> değeri verilir.
TABLE_SCHEMA	Şema adı. Yine provider'ımız şemaları desteklemiyorsa <code>null</code> değeri verilir.
TABLE_NAME	Belirli bir tabloya ait şema bilgileri kullanılacaksa, örneğin bu tabloya ait sütun bilgileri alınacaksa kullanılır.

TABLE_TYPE	Veritabanından hangi tipteki tabloları alacağımızı belirtmek için kullanılır.

Tablo 2. OleDbSchemaGuid.Tables İçin Sınırlama Parametresi Elemanları

Ancak farzedelimki sadece kullanıcı tanımlı tabloların listesine ihtiyacımız var. İşte bu durumda sınırlandırıcı parametremiz devreye girecek. Bunun için, sınırlandırıcı parametremizin son elemanının değerini belirlememiz yeterli olacak. O halde bu işlemi nasıl gerçekleştirebiliriz? Bunun için uygulamamızın kodlarını aşağıdaki şekilde değiştirmemiz yeterlidir.

```
object[] objSinirlama;
objSinirlama=new object[]{null,null,null,"TABLE"};
tblTabloListesi=con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, objSinirlama);
```



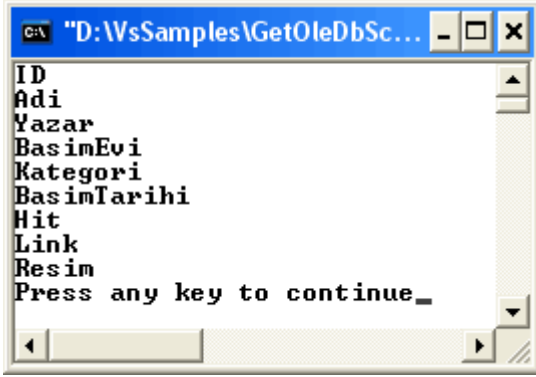
Şekil 2. Sadece Kullanıcı Tanımlı Tabloların Listesi

Burada sınırlandırıcı nesnemizin nasıl tanımlandığına dikkat edin. Bu tanımlama biraz karışık gelebilir. Ancak OleDbSchemaGuid özelliğinin aldığı değere göre şema bilgileri üzerinde bir sınırlama koymak istiyorsak, bu üyenin kullanabileceği değerleri bilmemiz gerekir. Burada object dizimizin son elemanın TABLE değeri atanmıştır. TABLE değeri şema bilgisine sadece kullanıcı tarafından tanımlanmış tabloların alınacağını belirtir. Diğer yandan bu dördüncü elemana, ALIAS, SYSTEM TABLE, VIEW, SYSTEM VIEW, SYNONYM, TEMPORARY, LOCAL TEMPORARY değerleride verebiliriz. Hepsi, OleDbSchemaGuid.Tables parametresi sonucunda oluşturulacak tablo şema bilgilerinin yapısınıda değiştirecektir. Örneğin, SYSTEM TABLE değerini vermemiz halinde, Friends tablosundaki system tablolarının listesini elde etmiş oluruz. Yada VIEW değerini verdiğimizde kullanıcı tanımlı görünüm nesnelerinin listesini elde etmiş olurduk.

Yukarıdaki örneğimizde, veritabanımızda yer alan tablolara ait schema bilgilerini aldık. Şimdi ise belirli bir tablonun sütun bilgilerini almak istediğimizi farzedelim. Burada sütun bilgileri için, OleDbSchemaGuid.Columns parametresini kullanmamız gerekiyor. Bu durumda sınırlandırıcımızda bu parametreye göre değiştirmek durumundayız. Eğer ilk örneğimizdeki gibi null değerini verirsek veritabanındaki tüm tabloların kolonlarına ait şema bilgilerini elde etmiş oluruz. Oysaki belirli bir tabloya ait alanlar için şema bilgisi alıcağsak,OleDbSchemaGuid.Columns parametresinin sınırlandırıcı koşullarını değiştirmemiz gerekecektir. İşte bu noktada sınırlandırıcı değişkenimizin üçüncü elemanı olan TABLE_NAME elemanını kullanırız. Bu amaçla kodlarımızı aşağıdaki şekilde değiştirmemiz yeterli olacaktır.

```
object[] objSinirlama;
objSinirlama=new object[]{null,null,"Kitap",null};
```

```
tblTabloListesi=con.GetOleDbSchemaTable(OleDbSchemaGuid.Columns, objSinirlama); /* OleDbConnection nesnemizin,
GetOleDbSchemaTable metodunu çalıştırıyoruz. Bu kez, Columns değeri sayesinde, sınırlama nesnemizin üçüncü elemanında
belirttiğimiz tablo adına ait alanların listesini elde ediyoruz. */
foreach(DataRow dr in tblTabloListesi.Rows)
{
    Console.WriteLine(dr["COLUMN_NAME"]);
}
```



Şekil 3. Tablomuzun ait alanların adları.

Böylece Kitap isimli tablomuzun alanlarına ait schema bilgilerini elde etmiş olduk. Bu örnekte ve bir önceki örnekte dikkat ederseniz, DataTable nesnesindeki belirli bir alanın değerini ekrana yazdırdık. Bu örnekte, COLUMN_NAME, önceki örnekte ise, TABLE_NAME. Elbette elde ettiğimiz şema bilgilerinde sadece bu alanlar yer almıyor. Örneğin Friends veritabanındaki kullanıcı tanımlı tablolara ait başka ne tür bilgilerin şemaya alındığına bir bakalım. Bu amaçla, DataTable nesnemize aktarılan şema bilgilerine ait satırlardaki tüm alanları bir döngü ile gezmemiz yeterli olacaktır. Bunu aşağıdaki kodlar ile gerçekleştirebiliriz.

```
object[] objSinirlama;
objSinirlama=new object[]{null,null,null,"TABLE"};
tblTabloListesi=con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, objSinirlama);
foreach(DataRow dr in tblTabloListesi.Rows)
{
    for(int i=0;i<=tblTabloListesi.Rows.Count;++i)
    {
        Console.Write(dr[i]);
        Console.Write("---");
    }
    Console.WriteLine("");
}
```

Veritabanı	Yaratan	Tablo Adı	Tablo Tipi (Burada Kullanıcı Tanımlı Olduğu Belirtiliyor)	Oluşturulma Tarihi
Friends	dbo	Sayac	TABLE	28.01.2004 17:04:17
Friends	dbo	Sepetler	TABLE	09.02.2004 17:24:39
Friends	dbo	Siparis	TABLE	28.01.2004 17:04:17
Friends	dbo	Siparisler	TABLE	09.02.2004 17:26:32
Friends	dbo	Siteler	TABLE	28.01.2004 17:04:18

Şekil 4. Şemadaki diğer bilgiler.

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

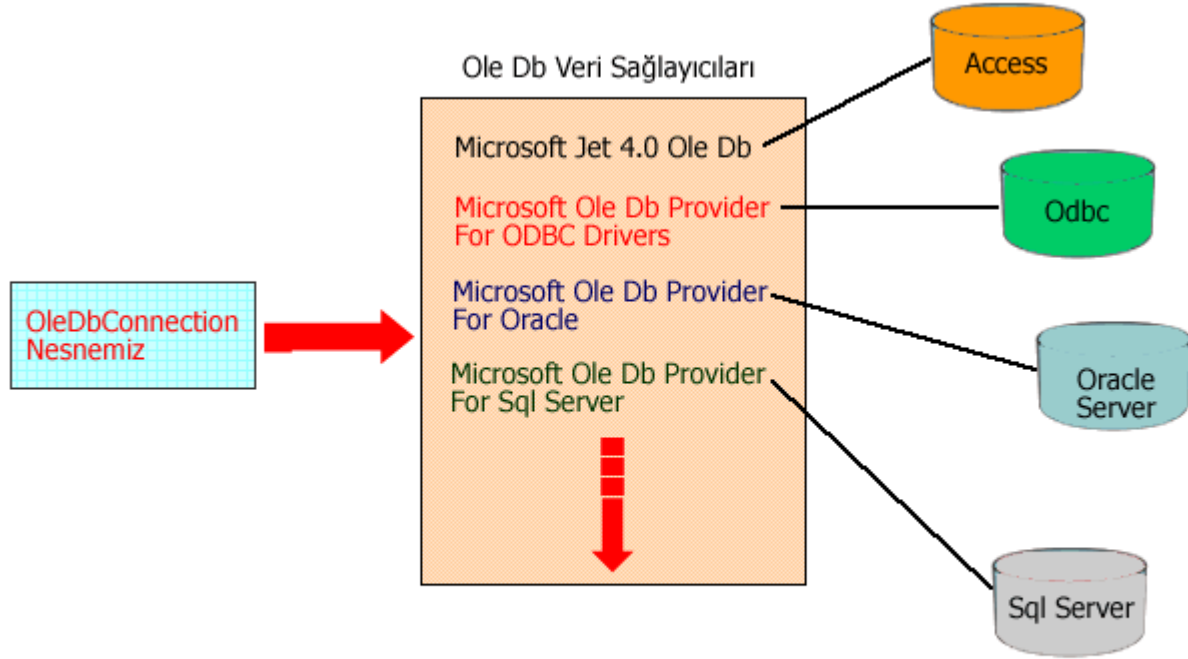
Connection (Bağlantı) Kavramı ve OleDbConnection Sınıfı

Değerli Okurlarım, Merhabalar.

Bu makalemizde, ADO.NET mimarisinde temel yapı taşı olan Connection (Bağlantı) kavramına kısaca değinecek ve OleDbConnection sınıfını incelemeye çalışacağız. ADO.NET mimarisinde, pek çok sınıfın veri kaynakları ile olan iletişimde Connection (Bağlantı) nesnelerini kullanırız. Örneğin, bir veri kaynağındaki tablolara ait verileri, DataSet sınıfından bir nesne örneğine taşımak istediğimizi düşünelim. Bu DataSet nesnesini dolduracak olan DataAdapter sınıfına, sahip olduğu sql sorgusunun veya komutunun işleyeceği bir hattı belirtmemiz gerekir. İşte burada devreye Connection (Bağlantı) nesnelerimiz girer. Yada bir Command sınıfı nesnesi yardımıyla veritabanı üzerindeki bir saklı yordamı (stored procedure) çalıştırmak istediğimizi düşünelim. Bu durumda komutun çalıştırılabileceği bir hattı veri kaynağımız ile Command nesnesi arasında sağlamamız gerekir. İşte Connection (Bağlantı) nesnemizi kullanmamız için bir sebep daha.

Verdiğimiz bu basit örneklerdende anlaşıldığı gibi, Connection(bağlantı) sınıfları, veri kaynağına bir hat çekerek, ADO.NET nesnelerinin bu hat yardımıyla işlemlerini gerçekleştirmelerine imkan sağlarlar. Ancak sahip olunan veri kaynağının türüne göre, ADO.NET içerisine değişik Connection sınıfları eklenmiştir. DotNet'in ilk sürümünde OleDbConnection ve SqlConnection nesneleri ile bu hatlar temin edilirken, .NET Framework 1.1 sürümü ile birlikte, OdbcConnection ve OracleConnection sınıfları da ADO.NET kütüphanelerine dahil edilerek Odbc ve Oracle veri kaynaklarına bağlantılar sağlanması imkanı kazandırılmıştır.

OleDbConnection sınıfı ile, bir OleDb Data Provider (veri sağlayıcısı) üzerinden, ole db destekli veri kaynaklarına erişim sağlayabiliriz. SqlConnection sınıfı Sql Sunucularına doğrudan bağlantı sağlar. Aynı şekilde OracleConnection sınıfı da Oracle veri kaynaklarına doğrudan erişim sağlar. OdbcConnection sınıfı ise odbc destekli veri kaynaklarına erişim için kullanılır. Bu makalemizde bu bağlantı sınıflarından OleDbConnection sınıfını inceleyeceğiz. OleDbConnection sınıfı, ADO.NET sınıflarının, Ole Db desteği olan veri kaynaklarına erişebilmesi amacıyla kullanılır. Veri kaynağının tipi tam olarak bilinmediği için, arada bu işlevi ayırt etmeye yarayan bir COM+ nesnesi yer almaktadır. OleDbConnection sınıfına ait bir nesne iletişim kurmak istediği veri kaynağına ait ole db veri sağlayıcısını belirtmek durumundadır. Bunu daha iyi kavramak için aşağıdaki şekle bakalım.



Şekil 1. OleDbConnection ile Veri Kaynaklarına Bağlantı.

Görüldüğü gibi bir OleDbConnection nesnesi öncelikle bir Ole Db Data Provider (Ole Db Veri Sağlayıcısı) ile iletişim kurar. Ardından bu veri sağlayıcı istenen veri kaynağına erişerek, gerekli hattı tesis etmiş olur. Peki bu işlemi nasıl gerçekleştireceğiz. İşte tüm Connection nesnelerinin en önemli özelliği olanConnectionString özelliği bu noktada devreye girmektedir. Kısaca ConnectionString özelliği ile, veri kaynağı ile sağlanacak olan iletişim hattının kurulum bilgileri belirlenir. OleDbConnection sınıfı için ConnectionString özelliği aşağıdaki prototipe sahiptir.

```
public virtual string ConnectionString {get; set;}
```

ConnectionString özelliği belirlenmiş bir OleDbConnection sınıfı nesne örneğini açtığımızda, yani veri kaynağına olan hattı kullanılabilir hale getirdiğimizde, bu özellik yalnız-okunabilir (read-only) hale gelir. Dolayısıyla açık bir OleDbConnection nesnesinin ConnectionString özelliğini değiştiremezsiniz. Bunun için bu bağlantıyı tekrardan kapatmanız gerekecektir. ConnectionString özelliği, bir takım anahtar-değer çiftlerinin noktalı virgül ile ayrılmasından oluşturulan string bir bilgi topluluğudur. ConnectionString özelliği içinde kullanabileceğimiz bu anahtar-değer çiftlerinin en önemlisi Provider anahtarıdır. Bu anahtara vereceğimiz değer, hangi tip ole db veri sağlayıcısını kullanmak istediğimizi belirtmektedir.

Örneğin Sql sunucusuna, Sql Ole Db Provider ile bağlanmak istersek, Provider anahtarına, SQLOLEDB değerini atarız. Provider anahtarı mutlaka belirtilir. Daha sonraki anahtar-değer çiftleri ise bu Provider seçimine bağlı olarak değişiklik gösterecektir. Veri kaynağına hangi tip Ole Db Veri Sağlayıcısından bağlandığımızı seçtikten sonra, bağlanmak istediğimiz veri kaynağıda belli olmuş olacaktır. Sırada bu veri kaynağının adını veya adresine belirteceğimiz anahtar-değer çiftlerinin belirlenmesi vardır. Örneğin bir Sql Sunucusuna bağlanıyorsak, sunucu adında Ole Db Data Provider(Veri Sağlayıcısı) 'na bildirmemiz gerekir. Bunun için Data Source

anahtarını kullanırız. Bununla birlikte bağlandığımız veri kaynağı, sql yada oracle gibi bir veritabanı yönetim sistemi değilse, Access gibi bir tablolama sistemi ise, Data Source anahtarı, tablonun fiziki adresini alır. Sql ve Oracle gibi sunuculara yapılacak bağlantılarda Provider ve Data Source seçiminin yanında hangi veritabanına bağlanılacağına Initial Catalog anahtarı yada Database anahtarı ile belirleriz. Bunların dışında veri kaynağına yapılacak olan bağlantının güvenlik ayarlarının da belirtiriz. Çoğunlukla Integrated Security gibi bir anahtara True değerinin atandığını görürüz. Bu anahtar, veri kaynağına bağlanmak istenen uygulama için, makinenin windows authentication ayarlarına bakıldığını belirtir. Dolayısıyla sql sunucusuna bağlanma yetkisi olan her windows kullanıcısı bu bağlantıyı sağlayabilir. Ancak istersek belli bir kullanıcı adı veya şifresi ile de bir veritabanına bağlantı açılmasını sağlayabiliriz. Bunun için ise, User ID ve Password anahtarlarını kullanırız.

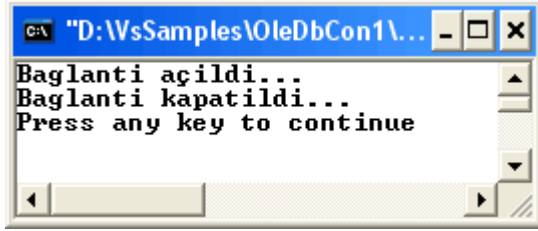
Buraya kadar bahsettiklerimiz kavramsal açıklamalardır. Dilerseniz basit örnekler ile konuyu daha iyi açıklamaya çalışalım. Örneklerimizi Console uygulamaları şeklinde gerçekleştireceğiz. İlk örneğimizde, Sql Sunucusundaki veritabanı için, bir bağlantı hattı oluşturup açacağız.

```
using System;
using System.Data.OleDb; /* OleDbConnection sınıfı, Data.OleDb isim uzayında yer almaktadır. */
namespace OleDbCon1
{
    class Class1
    {
        static void Main(string[] args)
        {
            OleDbConnection conFriends=new OleDbConnection(); /* OleDbConnection nesnemiz oluşturuluyor. */
            /* ConnectionString özelliği belirleniyor. Provider (Sağlayıcımız) SQLOLEDB. Bu bir sql sunucusuna bağlanmak
            istediğimizi belirtir. Data Source anahtarına localhost değerini atayarak, sunucunun yerel makinede olduğunu belirtiyoruz. Ancak
            buraya başka bir adreste girilebilir. Sunucunuz nerede ise oranın adresi. Database ile, bağlantı hattının açılacağı veritabanını
            belirliyoruz. Burada sql sunucumuzda yer alan Friends veritabanına bağlantı hattı açıyoruz. Son olarak Integrated Security=SSPI
            anahtar-değer çifti sayesinde Windows Doğrulaması ile sunucuya bağlanabileceğimizi belirtiyoruz. Yani sql sunucusuna bağlanma
            yetkisi olan her windows kullanıcısı bu hattı tesis edebilecek.*/
            conFriends.ConnectionString="Provider=SQLOLEDB;Data Source=localhost;Database=Friends;Integrated
            Security=SSPI";
            try
            {
                conFriends.Open(); /* Open metodu ile oluşturduğumuz iletişim hattını kullanıma açıyoruz. */
                Console.WriteLine("Bağlantı açıldı...");
                conFriends.Close(); /* Close metodu ile de oluşturulan iletişim hattını kapatıyoruz. */
                Console.WriteLine("Bağlantı kapatıldı...");
            }
            catch(Exception hata)
            {
                Console.WriteLine(hata.Message.ToString());
            }
        }
    }
}
```

```

    }
}
}
}

```



Şekil 2. Uygulamanın Çalışmasının Sonucu.

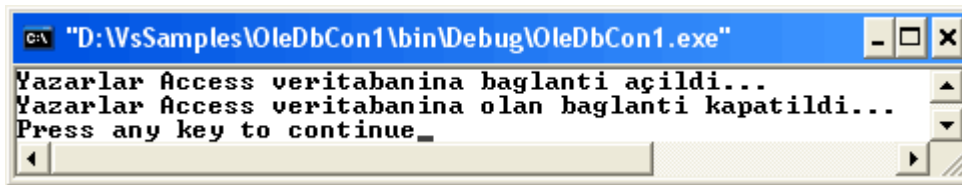
Aynı örnekte bu kez belli bir kullanıcı ile bağlanmak istediğimizi düşünelim. Bu durumda ConnectionString'imizi aşağıdaki şekilde değiştirmemiz gerekir. Bu durumda User ID ve Password anahtarlarına gerekli kullanıcı değerlerini atarız.

```
conFriends.ConnectionString="Provider=SQLOLEDB;Data Source=localhost;Database=Friends;User Id=sa;Password=CucP??80.";
```

Şimdide bir Access tablosuna nasıl bağlanabileceğimizi görelim. Bunun için ConnectionString özelliğimizi aşağıdaki gibi yazarız.

```
OleDbConnection conYazarlar=new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;data source=c:\\Authors.mdb");
try
{
    conYazarlar.Open();
    Console.WriteLine("Yazarlar Access veritabanına bağlantı açıldı...");
    conYazarlar.Close();
    Console.WriteLine("Yazarlar Access veritabanına olan bağlantı kapatıldı...");
}
catch(Exception hata)
{
    Console.WriteLine(hata.Message.ToString());
}

```



Şekil 3. Access Veritabanına Bağlatı.

Bu örnekte dikkat ederseniz ConnectionString özelliğini, OleDbConnection nesnemizin diğer yapıcı metodu içerisinde parametre olarak belirledik. Ayrıca, Provider olarak bu kez Microsoft.Jet.OLEDB.4.0 'ı seçerek, bir Access veritabanına bağlanmak istediğimizi

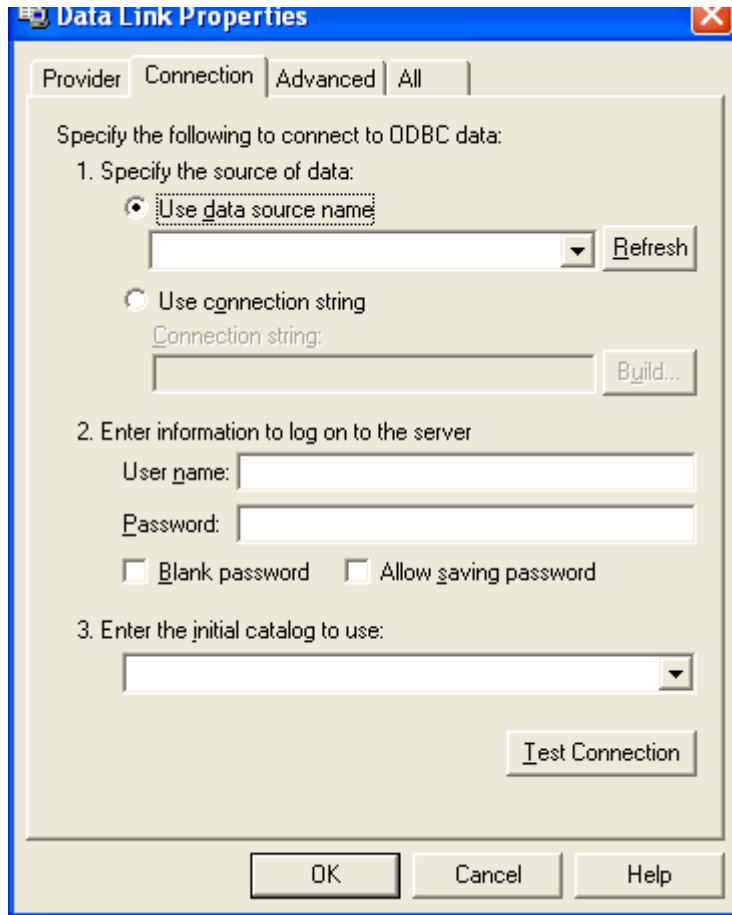
bu Ole Db Provider'a bildirmiş olduk. Daha sonra bu veri sağlayıcı componenti, Data Source anahtarındaki değere bakarak, ilgili adresteki veritabanına bir hat çekti.

Başka bir ConnectionString anahtarında File Name dir. Bu anahtara bir udl uzantılı dosya adresi vererek, bağlantı hattının bu dosyadaki ayarlar üzerinden gerçekleştirilmesini sağlayabiliriz. Bir udl dosyası Data Link Properties (veri linki özellikleri) özelliklerini belirler. Böyle bir dosya oluşturmak son derece basittir. Bir text editor ile boş bir dosya açın ve onu udl uzantısı ile kaydedin. Bu durumda dosyamızın görünümü şu şekilde olacaktır.



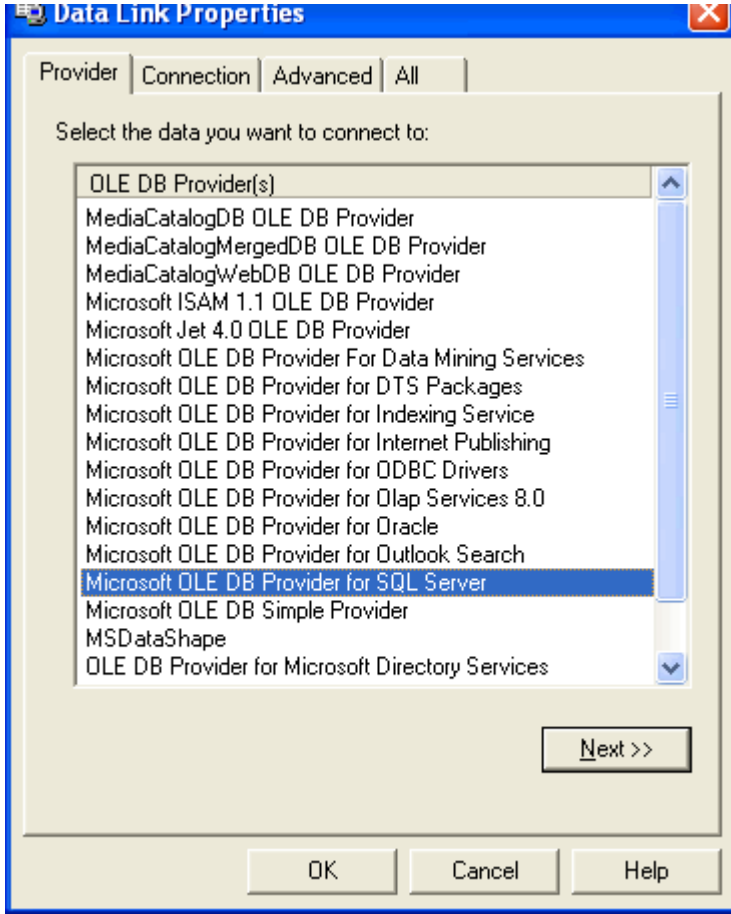
Şekil 4. Bir udl dosyasının görüntüsü.

Bu dosyayı açtığımızda hepimizin aşına olduğu veritabanı bağlantı seçenekleri ile karşılaşırız.



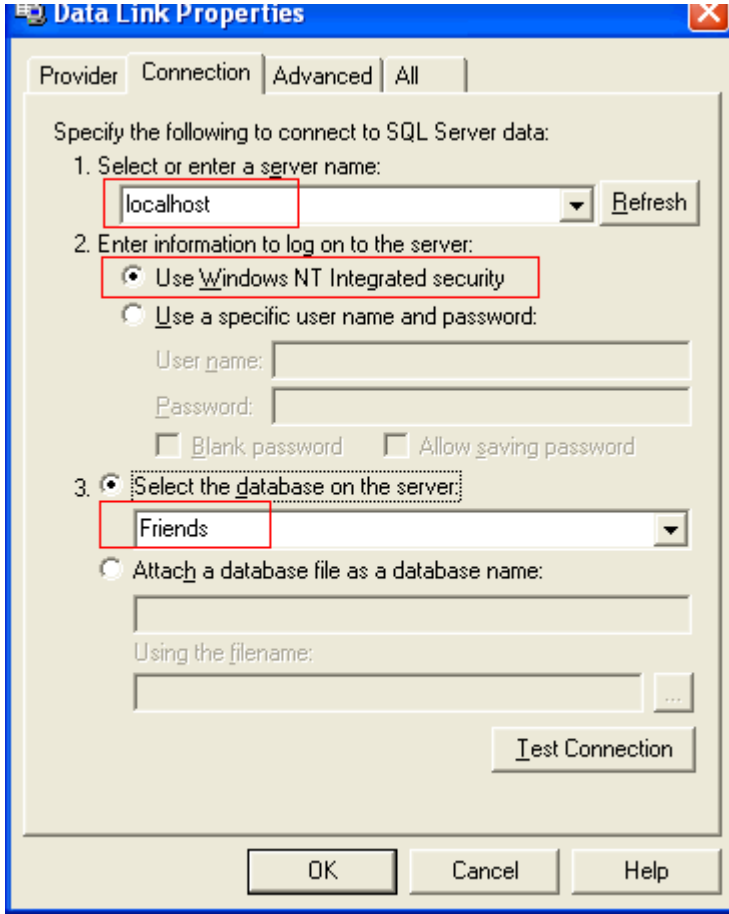
Şekil 5. Connection ayarları.

Bu penceredeki adımları takip ederek bir ConnectionString'i bir udl dosyasına kaydedebilir ve OleDbConnection nesnemiz için sadece File Name anahtarına bu değeri vererek ilgili bağlantının, bu udl dosyasındaki ayarlar üzerinden gerçekleştirilmesini sağlayabiliriz. İlk yapmamız gereken ConnectionString özelliğinde olduğu gibi , Provider (Sağlayıcı) seçimidir.



Şekil 6. Provider Seçimi.

Burada örnek olarak Sql Server Provider'ımızı seçelim. Sonraki adımda ise sırasıyla sunucu adımızı (Server Name), sunucuya hangi kimlik doğrulaması ile bağlanacağımızı ve veritabanımızın adını seçeriz. Son olarakta bu dosyamızı kaydedelim.



Şekil 7. Connection Özelliklerinin belirlenmesi.

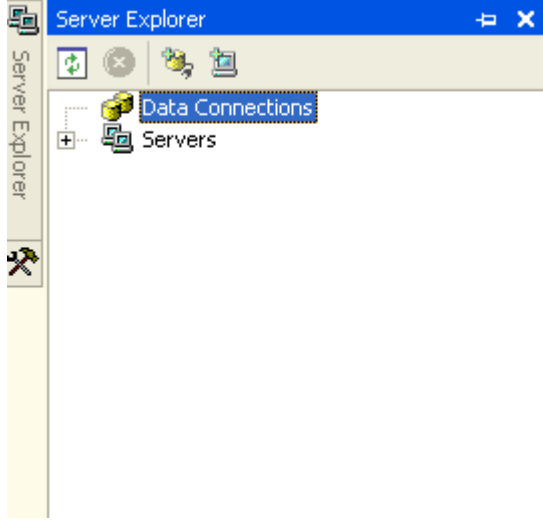
Şimdi uygulamızı buna göre değiştirelim.

```
OleDbConnection con=new OleDbConnection();
con.ConnectionString="File Name=C:\\baglanti.udl";
try
{
    con.Open();
    Console.WriteLine("Bağlantı açıldı...");
    con.Close();
    Console.WriteLine("Bağlantı kapatıldı...");
}
catch(Exception hata)
{
    Console.WriteLine(hata.Message.ToString());
}
```



Şekil 8. Udl dosyası üzerinden bağlantı açmak ve kapatmak.

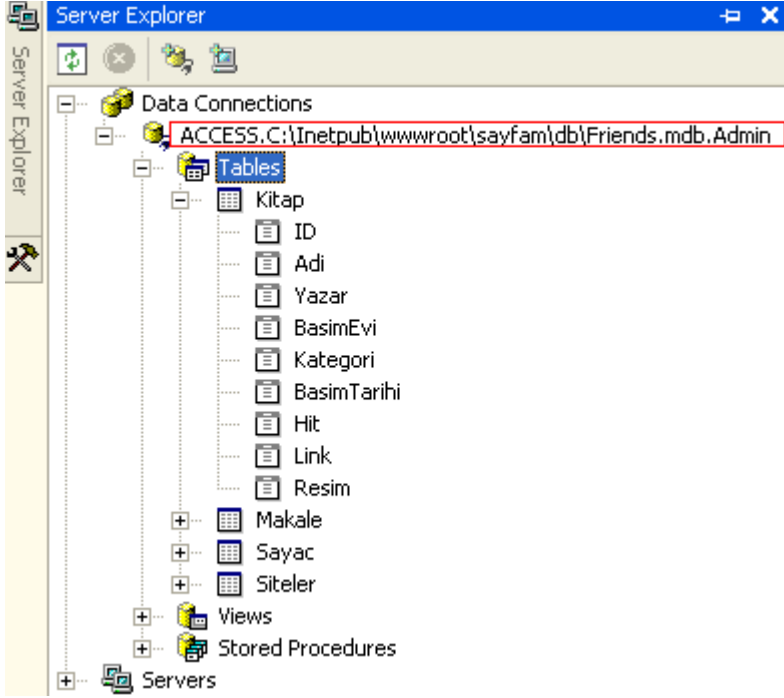
Tabiki burada Visual Studio.Net'in bağlantı sağlamak için bize sunduğu görsel nimetlerden söz etmede geçemeyiz. Visual Studio.NET ortamında, OleDbConnection oluşturmak için kullanabileceğimiz Server Explorer sekmesi yer almaktadır.



Şekil 9. Server Explorer

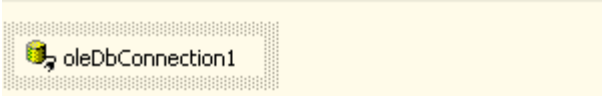
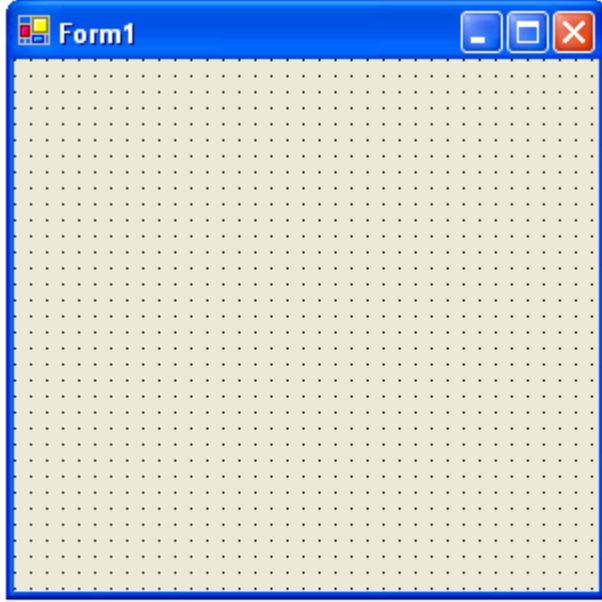
Burada servers sekmesinde Sql sunucularına erişebiliriz. Buradan tablolara, görünümlere, saklı yordamlara hatta tablo alanlarına ulaşabiliriz. Server Explorer'ın sağladığı pek çok işlevsellik vardır. Örneğin bu pencereden, bir sql tablosu yaratabilirsiniz veya bir saklı yordam oluşturabilirsiniz. Hatta bir tabloyu formunuza sürüklüyerek, gerekli olan tüm ADO.NET nesnelerinin otomatik olarak oluşturulmasını sağlayabilirsiniz. Bu makalemizde OleDbConnection nesnesini incelediğimiz için bu tip bir bağlantıyı Server

Explorer yardımıyla nasıl gerçekleştireceğimizi inceleyeceğiz. Bunun için  (Connect To Database) aracını kullanacağız. Bu butona bastığımızda karşımıza tanıdık Data Link Properties penceresi gelecektir. Süratli bir şekilde burada gerekli bağlantı ayarlarını yaptıktan sonra, Server Explorer aşağıdaki görünümü alacaktır. Burada örnek olarak bir Access veritabanına bağlantı sağlanmıştır.



Şekil 10. Bağlantı oluşturuldu.

Dikkat ederseniz bağlantımız üzerinden, veri kaynağındaki tablolara ve alanlarına ulaşabiliyoruz. Şimdi bu bağlantıyı, bir windows uygulamasında veya bir asp.net uygulamasında forma sürüklediğimizde bir OleDbConnection nesnesinin otomatik olarak oluşturulduğunu göreceksiniz. İşte bu kadar basit. Artık bu connection nesnesini kullanabilirsiniz. Diğer yandan Server Explorer'da oluşturulan bu bağlantıyı başka uygulamalarda da hazır olarak görebilirsiniz. Data Connection sekmesi uygulamalarda kullanacağımız veri bağlantılarını hazır olarak bulundurmak için ideal bir yerdir.



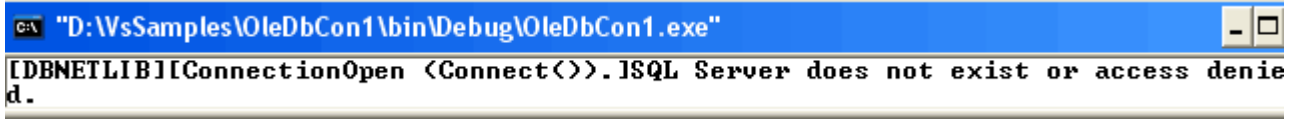
Şekil 11. OleDbConnection1 nesnesinin Server Explorer yarılımıyla oluşturulması.

Gelelim OleDbConnection nesnesinin diğer kullanışlı üyelerine. Buraya kadar bir bağlantı hattını oluşturmak için ConnectionString özelliğinin nasıl kullanıldığını inceledik. Bununla birlikte var olan bir bağlantı hattını açmak için Open metodunu, bu bağlantı hattını kapatmak içinde Close metodunun kullanıldığını gördük. OleDbConnection nesnesinin diğer bir özelliğide ConnectionTimeout değeridir. ConnectionTimeout özelliği, bir bağlantının sağlanması için gerekli süreyi belirtir. Bu süre boyunca bağlantı sağlanamaması bir istisnanın fırlatılmasına neden olucaktır. Bu özellik yalnız-okunabilir (read-only) bir özellik olduğundan, değerini doğrudan değiştiremeyiz. Bunun için, bu özelliği ConnectionString içerisinde belirlememiz gerekir. Örneğin aşağıdaki kodlarda, Sql sunucusuna bağlanabilmek için gerekli süre 10 saniye olarak belirlenmiştir. Şimdi ben Sql Server servisimizi durduracağım ve uygulamayı çalıştıracam. Bakalım 10 saniye sonunda ne olacak.

```
OleDbConnection conFriends=new OleDbConnection();
conFriends.ConnectionString="Provider=SQLOLEDB;Data Source=localhost;Database=Friends;User
Id=sa;Password=CucP??80.;Connect Timeout=10";
try
{
    conFriends.Open();
    Console.WriteLine("Baglanti açildi...");
    conFriends.Close();
    Console.WriteLine("Baglanti kapatildi...");
}
catch(Exception hata)
```

```
{
    Console.WriteLine(hata.Message.ToString());
}
```

Bu durumda aşağıdaki hata mesajını alırız.



Şekil 12. Sql sunucusunun olmadığını gösteren istisna.

OleDbConnection sınıfının Open ve Close metodları dışındada faydalı metodları vardır. Örneğin ChangeDatabase metodu. Bu metod ile açık olan bir bağlantı üzerinden, veri kaynağındaki seçili veritabanını değiştirmemiz sağlanır. Yani hattın ucu başka bir veritabanına yönlendirilebilir. Bu tabiki Oracle ve Sql gibi veritabanı sistemlerinde özellikle işe yarar. Örneğin, Friends veritabanına bağliyken, açık olan bağlantımız üzerinden hattımızı, pubs veritabanına yönlendirelim.

```
OleDbConnection conFr=new OleDbConnection(); /* OleDbConnection nesnemiz oluşturuluyor. */
conFr.ConnectionString="Provider=SQLOLEDB;Data Source=localhost;Database=Friends;Integrate/d Security=SSPI"; /* Bağlantı
hattımız için gerekli bilgiler giriliyor. Sql sunucumuzda yer alan Friends isimli veritabanına bağlandık. */
conFr.Open(); /* Bağlantımız açılıyor. */
Console.WriteLine("Veritabanı "+conFr.Database.ToString()); /* Şuandaki bağlantının hangi veritabanına yapıldığını
OleDbConnection sınıfının Database özelliği ile öğreniyoruz. */
conFr.ChangeDatabase("pubs"); /* ChangeDatabase metodu ile bağlantı hattımızı yönlendirmek istediğimiz veritabanının adını
giriyoruz. */
Console.WriteLine("Şimdiki veritabanı "+conFr.Database.ToString()); /* Bağlantı hattının şu an yönlendirilmiş olduğu veritabanının
adını Database özelliği ile elde ediyoruz.*/
conFr.Close(); /* Bağlantımızı kapatıyoruz. */
```



Şekil 13. ChangeDatabase metodunun çalışmasının sonucu.

Diğer yararlı bir metod ise GetOleDbSchemaTable metodudur ki bunu bir önceki makalemizde incelemiştik. Bunun dışında bir OleDbCommand nesnesini oluşturmaya yarayan CreateCommand metodu, bir Transaction'ın başlatılması için kullanılan BeginTransaction metodu, OleDbConnection'a ait kaynakları serbest bırakan Dispose metodu'da faydalı diğer metodlar olarak sayılabilir. Bu metodlardan ilerliyen makalelerde yeri geldikçe bahsedeceğiz. OleDbConnection nesnesinin sadece 3 adet olayı vardır. Bunlar, StateChange, Disposed ve InfoMessage olaylarıdır. Bunlardan en çok, StateChange olayını kullanırız. Bu olay, OleDbConnection nesnesinin bağlantı durumunda bir değişiklik olduğunda oluşur. Bu olayın prototipi aşağıdaki şekildedir.

public event StateChangeEventHandler **StateChange;**

Bu olay StateEventArgs tipinden bir argüman almaktadır. Bu argüman iki özelliğe sahiptir. Bunlar CurrentState ve OriginalState özellikleridir. CurrentState bağlantının o anki durumunu belirtir. OriginalState ise son değişiklikten önceki halini gösterir. Her iki özellikte ConnectionState numaralandırıcısı türünden değerlere işaret ederler. Bu değerler şunlardır.

ConnectionState Değeri	Açıklaması
ConnectionState.Open	Bağlantı açık ise bu değer geçerlidir.
ConnectionState.Closed	Bağlantı kapandığında bu değer geçerlidir.
ConnectionState.Connecting	Bağlantı hattı iletişime açılırken bu değer geçerlidir.
ConnectionState.Broken	Bağlantı hattı açıkken herhangi bir nedenle bir kopma meydana gelmesi ve hattın işlevselliğini kaybetmesi durumunda oluşur.
ConnectionState.Executing	Bağlantı nesnemiz bir komut çalıştırırken oluşur.
ConnectionState.Fetching	Bağlantı hattı üzerinden veriler alınırken bu değer geçerlidir.

Tablo 1. ConnectionState numaralandırıcısının değerleri.

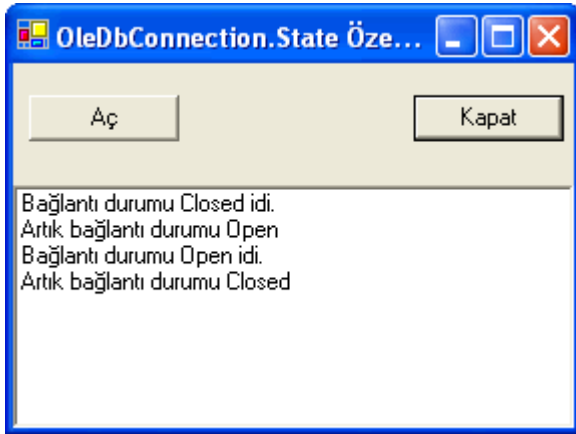
ConnectionState numaralandırıcısı aynı zamanda, State özelliği içinde kullanılabilir. State özelliği, OleDbConnection nesnesinin o anki durumunu, ConnectionState numaralandırıcısı tipinde saklar. State özelliğini uygulamamızda, var olan bağlantının durumun kontrol ederek hareket etmek için kullanabiliriz. Örneğin bir bağlantı nesnesini uygulamamızın bir yerinde tekrardan açmak istediğimizi varsayalım. Bu bağlantı nesnesinin durumu zaten Open ise yani açık ise, tekrardan açma işlemi uygulamamız gerekmez. Dilerseniz makalemizin sonunda StateChange olayına ilişkin bir örnek geliştirelim.

```
OleDbConnection con; /* OleDbConnection nesnemiz tanımlanıyor.*/
private void Form1_Load(object sender, System.EventArgs e)
{
    IstDurum.Items.Clear();
    con=new OleDbConnection("Provider=SQLOLEDB;Data source=localhost;initial catalog=Friends;Integrated Security=sspi"); /*
Bağlantı hattımız oluşturuluyor. */
    con.StateChange+=new StateChangeEventHandler(con_DurumDegisti); /* OleDbConnection nesnemiz için StateChange
olayımız ekleniyor. Olay meydana geldiğinde con_DurumDegisti isimli metod çalıştırılacak.*/
```

```

}
private void btnAc_Click(object sender, System.EventArgs e)
{
    if(con.State==ConnectionState.Closed) /* Kullanıcı açık olan bir bağlantı üzerinden tekrar bu butona basarak bir bağlantı açmak
    isterse bunun önüne geçmek için ilgili OleDbConnection nesnesinin durumuna bakıyoruz. Eğer con nesnesi kapalı ise, açılabilmesini
    sağlıyoruz.*/
    {
        con.Open(); /* Bağlantımız açılıyor. İşte bu anda StateChange olayı çalıştırılır.*/
    }
}
private void btnKapat_Click(object sender, System.EventArgs e)
{
    if(con.State==ConnectionState.Open) /* Eğer açık bir bağlantı varsa kapatma işlemini uyguluyoruz.*/
    {
        con.Close(); /* Bağlantı kapanıyor. StateChange olayı bir kez daha çalışır. */
    }
}
private void con_DurumDegisti(object sender,StateChangeEventArgs e)
{
    IstDurum.Items.Add("Bağlantı durumu "+e.OriginalState.ToString()+" idi."); /* Bağlantımızın hangi halde olduğunu alıyoruz.*/
    IstDurum.Items.Add("Artık bağlantı durumu "+e.CurrentState.ToString()); /* Ve bağlantımızın yeni halini alıyoruz.*/
}

```



Şekil 14. StateChange olayı.

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

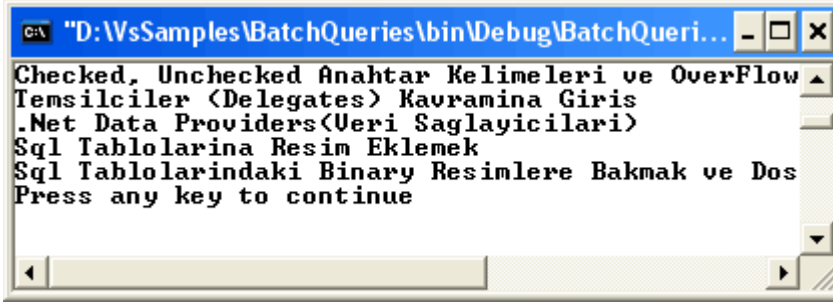
Batch Queries (Toplu Sorgular) ve SqlDataReader

Değerli Okurlarım, Merhabalar.

Bu makalemizde, toplu sorguların, SqlDataReader sınıfı ile nasıl okunabileceğini incelemeye çalışacağız. Bildiğiniz gibi SqlDataReader nesneleri, bir select sorgusunu çalıştıran SqlCommand sınıfına ait, ExecuteReader metodu ile oluşturulmaktaydı. Çalıştırılan sorgu sonucu elde edilen kayıt kümesinde sadece okunabilir ve ileri yönlü hareket etmemize imkan sağlayan SqlDataReader sınıfı, belli bir t anında veri kanağından sadece tek bir satırı okumamıza izin vermektedir. Bu yönden bakıldığında, SqlDataReader sınıfı, verileri hızlı ve verimli bir şekilde okumamıza imkan sağlamaktadır. Örneğin aşağıdaki kod satırları ile, Sql sunucumuzda yer alan makale isimli tablodaki tüm satırlar okunarak ekrana yazdırılmıştır.

```
using System;
using System.Data.SqlClient;
using System.Data;
namespace BatchQueries
{
    class Class1
    {
        static void Main(string[] args)
        {
            SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
/* Sql sunucumuza olan bağlantı hattını tesis edecek SqlConnection nesnemiz tanımlanıyor.*/
            SqlCommand cmdMakale=new SqlCommand("Select * From Makale",conFriends); /* Makale tablosundaki tüm satırları
alacak sql sorgusunu çalıştıracak SqlCommand nesnemiz oluşturuluyor. */
            SqlDataReader drMakale; /* SqlDataReader nesnemiz tanımlanıyor. */
            conFriends.Open(); /*Bağlantımız açılıyor. */
            drMakale=cmdMakale.ExecuteReader(CommandBehavior.CloseConnection); /* Komutumuz çalıştırılıyor ve sonuç
kümesinin başlangıcı SqlDataReader nesnemize aktarılıyor. */
            /* İleri yönlü olarak, SqlDataReader nesnemiz ile, sonuç kümesindeki satırlar okunuyor ve 1 indisli alanın değeri ekrana
yazdırılıyor. */
            while(drMakale.Read())
            {
                Console.WriteLine(drMakale[1]);
            }
            drMakale.Close(); /* Bağlantımız kapatılıyor. */
        }
    }
}
```

Bu kodları çalıştırdığımızda karşımıza aşağıdakine benzer bir sonuç çıkacaktır.



Şekil 1. Programın Çalışmasının Sonucu.

SqlDataReader nesnesinin kullanımına kısaca değindikten sonra makelemizin asıl konusu olan toplu sorgulara değinelim. Toplu sorgular birbirlerinden noktalı virgül ile ayrılmış sorgulardır. Birden fazla sorguyu bu şekilde bir araya getirerek işlemlerin tek bir hamlede başlatılmasını ve gerçekleştirilmesini sağlamış oluruz. Bu sorgu topluluklarını, eski dostumuz Ms-Dos işletim sistemindeki bat uzantılı Batch dosyalarına benzetebilirsiniz. Sorun şu ki, yukarıdaki tarzda bir SqlDataReader kullanımını bir toplu sorguya uyguladığımızda, sadece ilk sorgunun çalıştırılacak olmasıdır. Örneğin aşağıdaki biri bir toplu sorgumuz olduğunu düşünelim;

```
Select * From Makale;Select * From Kitap;Select * From Siteler
```

Bu toplu sorguda arka arkaya üç select sorgusu yer almaktadır. Makale, Kitap ve Siteler tablolarının tüm sütunları talep edilmektedir. Yukarıdaki kod tekniğini böyle bir toplu sorguya uyguladığımızı düşünelim.

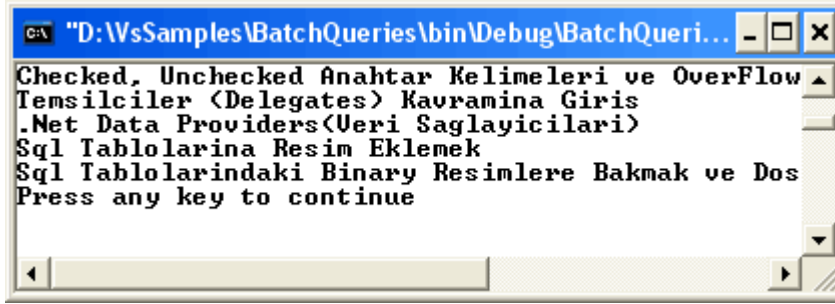
```
using System;
using System.Data.SqlClient;
using System.Data;
namespace BatchQueries
{
    class Class1
    {
        static void Main(string[] args)
        {
            SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
            string sorgu="Select * From Makale;Select * From Kitap;Select * From Siteler";
            SqlCommand cmdMakale=new SqlCommand(sorgu,conFriends);
            SqlDataReader drMakale;
            conFriends.Open();
            drMakale=cmdMakale.ExecuteReader(CommandBehavior.CloseConnection);
            while(drMakale.Read())
            {
                Console.WriteLine(drMakale[1]);
            }
        }
    }
}
```

```

        drMakale.Close();
    }
}
}

```

Uygulamamızı çalıştırdığımıza sadece Makale isimli tabloya ait verileri okuyabildiğimizi ve ekrana yazdırıldıklarını görürüz.



Şekil 2. Sadece ilk kayıt kümesi okundu.

Problem şudur. Toplu sorgumuz, birbirinden farklı üç kayıt kümesi getirmektedir. Bu nedenle, her bir kayıt kümesinin ayrı ayrı okunması gereklidir. Bunu gerçekleştirmek için ise, SqlDataReader nesnesini, Read metodu false değerini döndürdükten, yani geçerli kayıt kümesindeki tüm satırların okunması bittikten sonra, başka kayıt kümesinin olup olmadığı kontrol edilmelidir. Bize bu imkanı aşağıda prototipi verilen, NextResult metodu sağlamaktadır.

```
public virtual bool NextResult();
```

Bu metod geriye bool tipinde bir değer döndürür. Eğer güncel kayıt kümesinin okunması bittikten sonra başka bir kayıt kümesi var ise, true değerini döndürecektir. Bu durumda, toplu sorgularda bir sonraki kayıt kümesinin var olup olmadığını belirlemek için başka bir while döngüsünü kullanırız. İşte yukarıdaki toplu sorgumuz sonucu elde edilen tüm kayıt kümelerini okuyabileceğimiz kodlar;

```

using System;
using System.Data.SqlClient;
using System.Data;
namespace BatchQueries
{
    class Class1
    {
        static void Main(string[] args)
        {
            SqlConnection conFriends=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=sspi");
            string sorgu="Select * From Makale;Select * From Kitap;Select * From Siteler";
            SqlCommand cmdMakale=new SqlCommand(sorgu,conFriends);
            SqlDataReader drMakale;

```

```
conFriends.Open();
drMakale=cmdMakale.ExecuteReader(CommandBehavior.CloseConnection);
do
{
    Console.WriteLine("-----");
    while(drMakale.Read())
    {
        Console.WriteLine(drMakale[1]);
    }
    Console.WriteLine("-----");
}
while(drMakale.NextResult());
drMakale.Close();
}
}
```

Burada do while döngümüz bizim anahtar noktamızdır. While do döngüsü içinde, o an geçerli olan kayıt kümesindeki tüm satırlar okunur. Okunacak satırlar bittikten sonra, sqlDataReader nesnemize ait Read metodu false değerini döndürür ve bu while-do döngüsü sonra erer. Bu işlemin ardından do-while döngüsünde yer alan NextResult metodu çalıştırılır. Eğer arkada başka bir kayıt kümesi varsa, while-do döngümüz bu kayıt kümesi için çalıştırılır. Do-while döngümüz, tekniği açısından en az bir kere çalıştırılır. Zaten ExecuteReader metodu sonucu dönen toplu kayıt kümeleri söz konusu olduğunda, SqlDataReader nesnemiz okumak üzere hemen ilk kayıt kümesine konumlanır. İşte sonuç;

```

C:\ "D:\VsSamples\BatchQueries\bin\Debug\BatchQueries.exe"
.Net Data Providers<Veri Sağlayıcıları>
Sql Tablolarına Resim Ekleme
Sql Tablolarındaki Binary Resimlere Bakmak ve Dosya Olarak Kayd
-----
Adım Adım ADO.NET
Adım Adım Visual C#.NET
Adım Adım Web Veritabanı Geliştirme .NET Sürümü
.NET Başlangıç Düzeyi 70-305 ve 70-306 Sınavlarına Hazırlık
C# ile Veritabanı Programlama ve ADO.NET
C# Temel Başlangıç Kılavuzu
.NET Orta Düzey 70-305 ve 70-306 Sınavlarına Hazırlık
Adım Adım Microsoft ASP.NET
ASP.NET
Adım Adım Visual Basic.NET
ASP.NET ve C#
C# Herkes İçin
C# Programlama Kılavuzu
C#'i Kavramak
Visual Basic.NET Programlama Kılavuzu
Visual Basic.NET Uygulama Geliştirme Kılavuzu
-----
C # Nedir?
Microsoft .NET Akademik
Asp Nedir
Yaz Gelistir
-----
Press any key to continue_

```

Şekil 3. Toplu sorgunun çalıştırılması sonucu.

Böylece, SqlDataReader nesnesi ile, toplu sorguların çalıştırılması sonucu elde edilen kayıt kümeleri üzerinden nasıl veri okuyabileceğimizi incelemiş olduk. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Command Kavramı ve OleDbCommand Sınıfı

Değerli Okurlarım, Merhabalar.

Bu makalemizde, ADO.NET mimarisi içinde çok önemli bir yere sahip olan Command kavramını ve OleDbCommand sınıfına ait en temel üyeleri incelemeye çalışacağız. Veritabanı uygulamaları geliştiren her programcı mutlaka, veri kaynağına doğru bir takım sorgu komutlarına ihtiyaç duymaktadır. Örneğin, veri kaynağındaki bir tabloya yeni bir satır eklemek için, veri kaynağı üzerinde bir tablo yaratmak için veya veri kaynağından belli şartlara uyan veri kümelerini çekmek için vb... Tüm bu işlemler için ADO.NET mimarisi bize, sql sorgularını barındırabileceğimiz ve geçerli bir bağlantı hattı üzerinden çalıştırabileceğimiz Command sınıfını sunmaktadır. Şu an itibarıyla, ADO.NET mimarisi 4 temel Command sınıfı içerir. Bunlar, OleDbCommand, SqlCommand, OracleCommand ve OdbcCommand sınıflarıdır.

Bahsetmiş olduğumuz bu 4 Command sınıfıda temelde aynı görevler için tasarlanmışlardır. Farklılık sadece işaret ettikleri veri kaynaklarından ibarettir. Hepsinin görevleri ortaktır. Kullanıldıkları veri kaynakları üzerinde sql ifadelerinden oluşturulan komutları çalıştırmak. Bunun için elbette ihtiyacımız olan en önemli kaynak geçerli bir bağlantı hattının ve bu hattın eriştiği veri kaynağı için geçerli bir sql ifadesinin olmasıdır. Command sınıfları, çalıştıracakları sql ifadelerini temsil eden nesneler oldukları için Command (Komut) terimini bünyelerinde barındırırlar.

Bu makalemizde, Command sınıflarından OleDbCommand sınıfını incelemeye çalışacağız. OleDbCommand sınıfı, OleDb veri sağlayıcısı tarafından erişilebilen kaynaklar üzerinde sql ifadelerini çalıştırmamıza izin verir. OleDbCommand sınıfına ait nesne örneğini oluşturmak için kullanabileceğimiz üç farklı yol vardır. Bunlardan ilki OleDbCommand nesnesini new yapılandırıcısı ile her hangibir komut sözcüğü içermeyen oluşturur. Bu teknik için kullanılan yapıcı metodun prototipi aşağıdaki gibidir.

public OleDbCommand();

Bu teknik ile oluşturulan bir komut nesnesi için bir takım özellikleri sonradan belirleyebiliriz. Öncelikle geçerli bir bağlantı nesnesine yani bir OleDbConnection nesnesine ihtiyacımız vardır. Diğer gereklilik ise, OleDbCommand sınıfı nesne örneğinin, çalıştıracağı sql ifadesidir. Bu amaçlar için OleDbCommand sınıfının Connection ve CommandText özelliklerini kullanırız. Yukarıdaki teknik ile oluşturduğumuz bir OleDbCommand nesnesi için Connection özelliği null, CommandText özelliği ise boş bir string'e işaret etmektedir. Bu tekniği daha iyi anlamak için basit bir örnek geliştirelim. Örneğin, Sql Sunucumuzda yer alan Friends isimli veri tabanındaki taboya bir satır veri gireceğimiz aşağıdaki sql ifadesini çalıştıracak bir komut tasarlamak istediğimizi varsayalım.

Insert Into Siteler (Baslik,Adres,Resim,Icerik) Values('C#','www.csharpneder.com','images/resim1.jpg','C# üzerine her türlü makale.')

Bunun için yazacağımız kodlar aşağıdadır.

```
using System;
using System.Data.OleDb; /* OleDbCommand sınıfı bu isim uzayında yer almaktadır. */
namespace OleDbCmd1
{
    class Class1
    {
        static void Main(string[] args)
        {
            /* Önce geçerli bir bağlantı hattı oluşturmamız gerekiyor. */
            OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated security=sspi");
            OleDbCommand cmd=new OleDbCommand(); /* Komut nesnemiz oluşturuluyor. */
            cmd.Connection=con; /* Komut nesnesinin hangi bağlantı hattını kullanacağı belirleniyor. */
            cmd.CommandText="Insert Into Siteler (Baslik,Adres,Resim,Icerik) Values ('C#','www.csharpneder.com','images/resim1.jpg','C# üzerine her türlü makale.');" /* Komutun çalıştıracağı sql ifadesi belirleniyor. */
        }
    }
}
```

```
}
```

Görüldüğü gibi OleDbCommand sınıfını oluşturduktan sonra Connection ve CommandText özellikleri belirleniyor. Diğer yandan, bir OleDbCommand nesnesini aşağıda prototipi olan diğer yapıcı metodu ilede oluşturabiliriz.

```
public OleDbCommand(string cmdText, OleDbConnection connection);
```

Bu yapıcı metodumuz parametre olarak sql ifadesini string veri tipinde ve bağlantı nesnesinde OleDbConnection sınıfı tipinde almaktadır. Bu haliyle yukarıda yazdığımız kodları dahada kısaltabiliriz.

```
using System;
using System.Data.OleDb; /* OleDbCommand sınıfı bu isim uzayında yer almaktadır. */
namespace OleDbCmd1
{
    class Class1
    {
        static void Main(string[] args)
        {
            /* Önce geçerli bir bağlantı hattı oluşturmamız gerekiyor. */
            OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated
security=sspi");
            OleDbCommand cmd=new OleDbCommand("Insert Into Siteler (Baslik,Adres,Resim,Icerik)
Values('C#','www.csharpnedir.com','images/resim1.jpg','C# üzerine her türlü makale.')" ,con);
            .
            .
            .
        }
    }
}
```

OleDbCommand nesnesinin oluşturmak için bahsettiğimiz bu iki yol dışında kullanabileceğimiz iki aşırı yüklenmiş metod daha vardır. Bunların protoipleride aşağıdaki gibidir.

```
public OleDbCommand(string cmdText);
```

Bu prototip sadece sql ifadesini almaktadır. Komut nesnesine ait diğer özellikler (Connection gibi) sonradan belirlenir.

```
public OleDbCommand(string cmdText, OleDbConnection connection, OleDbTransaction transaction);
```

Bu prototip ise, komut ifadesi ve bağlantı nesnesi haricinde birde OleDbTransaction nesnesi tipinden bir parametre alır. Bu prototip çoğunlukla, bir iş parçacığı içine alınmak istenen komut nesneleri için idealdir. Bir OleDbCommand nesnesi oluşturabilmek için kullanabileceğimiz son yol ise OleDbConnection sınıfına ait CreateCommand metodunun aşağıdaki örnekte olduğu gibi kullanılmasıdır.

```

using System;
using System.Data.OleDb; /* OleDbCommand sınıfı bu isim uzayında yer almaktadır. */
namespace OleDbCmd1
{
    class Class1
    {
        static void Main(string[] args)
        {
            /* Önce geçerli bir bağlantı hattı oluşturmamız gerekiyor. */
            OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated
security=sspi");
            OleDbCommand cmd=con.CreateCommand();
            cmd.CommandText="Insert Into Siteler (Baslik,Adres,Resim,Icerik) Values('C#','www.csharpneder.com','images/resim1.jpg','C#
üzerine her türlü makale.')";
            .
            .
            .
        }
    }
}

```

Bu teknikte, geçerli olan bağlantı nesnesinin tesis ettiği hat üzerinde çalışacak OleDbCommand sınıfı nesne örneği, CreateCommand metodu ile oluşturulmuştur. Bu adımdan sonra tek yapılması gereken CommandText özelliğine sql cümlecini atamak olacaktır.

Tüm bu teknikler bir OleDbCommand sınıfı nesne örneğini oluşturmak içindir. Ancak komutu çalıştırmak için henüz bir adım atmış değiliz. Bu haliye program kodlarımız derlense dahi hiç bir işe yaramıyacaktır. Nitekim, OleDbCommand nesnelerinin temsil ettiği sql cümleciklerini çalıştırmamız gerekmektedir. Bu amaçla kullanabileceğimiz üç OleDbCommand metodu vardır. Bunlar; ExecuteNonQuery, ExecuteReader ve ExecuteScalar metodlarıdır. Üç metodda farklı amaçlar ve performanslar için kullanılır. Bu amaçlar, çalıştırmak istediğimiz sql ifadesine göre değişiklik göstermektedir. Söz gelimi, yukarıdaki kod parçalarında yer alan sql ifadesi DDL (Data Defination Language- Veri Tanımlama Dili) komutlarından birisidir. Benzer şekilde update, delete sorgularında böyledir. Diğer taraftan DML (Data Manipulation Language- Veri İdare Dili) komutları dediğimiz Create Table, Alter Table gibi komutlarda mevcuttur. Bu iki kategoriye ait komutlar, etki komutları olarak adlandırılırlar. Hepsinin ortak özelliği geriye sonuç döndürmemeleridir. Tamamiyle veri kaynağı üzerinde bir takım sonuçların doğmasına yardımcı olurlar. İşte bu tip komut cümlecikleri için, ExecuteNonQuery metodu kullanılır. Bu metodun prototipi aşağıdaki gibidir.

public virtual int ExecuteNonQuery();

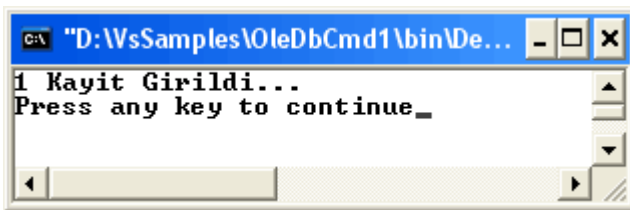
Bu metod görüldüğü gibi int veri tipinden bir tamsayıyı geri döndürür. Bu sayı komutun çalıştırılması sonucu etkilenen kayıt sayısını ifade etmektedir. Dolayısıyla bu metod, DDL ve DML komutları için geliştirilmiştir diyebiliriz. Örneğin, yukarıdaki kod parçalarını hayat geçirelim. Bunun için aşağıdaki kodları yazacağız.

```

using System;
using System.Data.OleDb; /* OleDbCommand sınıfı bu isim uzayında yer almaktadır. */
namespace OleDbCmd1
{
    class Class1
    {
        static void Main(string[] args)
        {
            /* Önce geçerli bir bağlantı hattı oluşturmamız gerekiyor. */
            OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated
security=sspi");
            OleDbCommand cmd=new OleDbCommand("Insert Into Siteler (Baslik,Adres,Resim,Icerik)
Values('C#','www.csharpnedir.com','images/resim1.jpg','C# üzerine her türlü makale.')" ,con);
            try
            {
                con.Open(); /* Bağlantımızı açıyoruz.*/
                int sonuc=cmd.ExecuteNonQuery(); /* Komutumuzu çalıştırıyoruz.ExecuteNonQuery metodunun döndüreceği değeri tam sayı
tipindeki sonuc değişkenine atıyoruz.*/
                Console.WriteLine(sonuc.ToString()+" Kayıt Girildi...");
            }
            catch(Exception hata)
            {
                Console.WriteLine(hata.Message.ToString());
            }
            finally
            {
                con.Close();
            }
        }
    }
}

```

Uygulamamızı çalıştırdığımızda aşağıdaki sonucu alırız.



Şekil 1. ExecuteNonQuery sonucu geri dönen değer.

Şimdi birde aşağıdaki örneğe bakalım. Bu kez elimizde, tablomuzun tüm satırlarındaki Resim alanlarının değerlerinin sonuna img ifadesini ekleyecek sql ifadesini içeren bir OleDbCommand nesnemiz olsun.

```
using System;
using System.Data.OleDb; /* OleDbCommand sınıfı bu isim uzayında yer almaktadır. */
namespace OleDbCmd1
{
    class Class1
    {
        static void Main(string[] args)
        {
            OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated
security=sspi");
            OleDbCommand cmdUpdate=new OleDbCommand("Update Siteler Set Resim=Resim+'img' ",con);
            try
            {
                con.Open();
                int Guncellenen=cmdUpdate.ExecuteNonQuery();
                Console.WriteLine(Guncellenen.ToString()+" Kayıt Güncellendi");
            }
            catch(Exception hata)
            {
                Console.WriteLine(hata.Message.ToString());
            }
            finally
            {
                con.Close();
            }
        }
    }
}
```

Bu durumda aşağıdaki sonucu alırız.

dbo.Siteler : T... (BURK1.Friends)		
	Adres	Resim
	http://www.csharpnedir.com	images/s1.jpgimg
demik	http://www.msakademik.net	images/s2.jpgimg
	http://www.aspnedir.com	images/s3.jpgimg
	http://www.yazgelistir.com	images/s4.jpgimg
	www.csharpnedir.com	images/resim1.jpgimg



Şekil 2. Update komutu sonucu ExecuteNonQuery'nin döndürdüğü değer.

Görüldüğü gibi tablomuzdaki 5 kaydın hepsi güncellenmiş ve ExecuteNonQuery geriye 5 değerini döndürmüştür. Bu çalıştırılan komut sonucu etkilenen kayıt sayısını öğrenmek için iyi ve etkili bir yoldur. ExecuteNonQuery metodu ile ilgili unutulmaması gereken nokta, bu metoden geriye her hangibir sonuç kümesi, her hangibir çıkış parametresi veya çıkış değeri döndürmemesidir. Elbette uygulamalarımızda, veri kaynaklarından veri kümeleri çekme ihtiyacını hissederiz. Böyle bir durumda ise, ExecuteReader metodunu kullanabiliriz. ExecuteReader metodu, çalıştırılan komut sonucu elde edilen sonuç kümesinden bir OleDbDataReader nesnesi için veri akışını sağlar. OleDbDataReader nesnesinin benzeri olan SqlDataReader nesnesi ve ExecuteReader metodunun kullanımını, SqlDataReader nesneleri ile ilgili makedelerimizde incelediğimiz için bu metodun nasıl kullanıldığını tekrar değinmiyorum.

OleDbCommand sınıfına ait bir diğer veri elde etme metodu ExecuteScalar metodudur. Prototipi aşağıdaki gibi olan bu metod sadece tek alanlık veri döndüren sql sorguları için kullanılır.

public virtual object ExecuteScalar();

Örneğin tablomuzdaki kayıt sayısının öğrenmek istiyoruz veya tablomuzdaki ücretler adlı alanda yer alan işçi ücretlerinin ortalamasının ne olduğunu bilmek istiyoruz yada primary key alanı üzerinden arama yaptığımız bir satıra ait tek bir sütunun değerini elde etmek istiyoruz. Bu tarz durumlarda, çalıştırılacak olan komut için bilgileri ExecuteReader metodu ile almak veya bilgileri bir DataSet kümesi içine almak vb... sistem kaynaklarının gereksiz yer harcanmasına ve performansın olumsuz şekilde etkilenerek azalmasına neden olur. Çare ExecuteScalar metodunu kullanmaktır. Örneğin;

```
using System;
using System.Data.OleDb; /* OleDbCommand sınıfı bu isim uzayında yer almaktadır. */
namespace OleDbCmd1
{
    class Class1
    {
        static void Main(string[] args)
        {
```

```

OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated
security=sspi");
OleDbCommand cmd=new OleDbCommand("Select Baslik From Siteler Where ID=8",con);
OleDbCommand cmdToplamSite=new OleDbCommand("Select Count(*) From Siteler",con);
try
{
con.Open();
Console.WriteLine("ID=8 olan satırın Baslik alanının değeri: "+cmd.ExecuteScalar().ToString());
Console.WriteLine("Site Sayısı: "+cmdToplamSite.ExecuteScalar().ToString());
}
catch(Exception hata)
{
Console.WriteLine(hata.Message.ToString());
}
finally
{
con.Close();
}
}
}

```

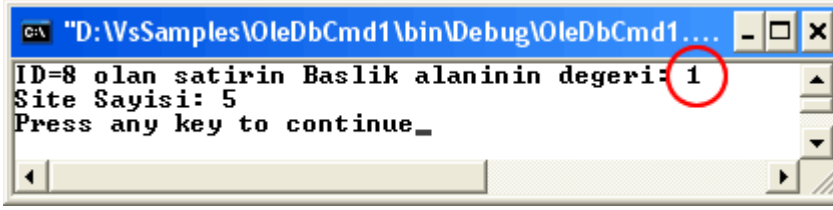


Şekil 3. ExecuteScalar Sonucu.

Bu örnekte, Siteler isimli tablomuza ID değeri 8 olan satırın sadece Baslik isimli alanının değerini veren bir komut nesnesi ve Siteler tabsundaki satır sayısını veren başka bir komut nesnesi kullanılmıştır. Her iki sql ifadeside tek bir hücreyi sonuç olarak döndürmektedir. Eğer sql ifadenizden birden fazla sütun alıyorsanız ve bu ifadeyi ExecuteScalar ile çalıştırıyorsanız, ilk satırın ilk sütunu haricindeki tüm veriler göz ardı edilecektir. Söz gelimi yukarıdaki örneğimizde, cmd OleDbCommand nesnesinin CommandText ifadesini aşağıdaki gibi değiştirelim.

```
OleDbCommand cmd=new OleDbCommand("Select * From Siteler",con);
```

Bu durumda aşağıdaki sonucu elde ederiz.



Şekil 4. ExecuteScalar sadece ilk hücreyi döndürür.

Görüldüğü gibi sonuç olarak, ilk satırın ilk alanının değeri elde edilmiştir. (ID alanının değeri.) OleDbCommand sınıfı ile veri kaynağında yer alan bir saklı yordamda (Stored Procedure) çalıştırabiliriz. Bu durumda CommandText olarak bu saklı yordamın adını girmemiz yeterli olacaktır. Ancak, çalıştırılacak olan komutun bir saklı yordamı çalıştıracağını belirtmemiz gerekmektedir. İşte bu noktada OleDbConnection sınıfı nesne örneğinin CommandType özelliğinin değerini belirtmemiz gerekir.

public virtual CommandType CommandType {get; set;}

Prototipi yukarıdaki gibi olan bu özellik, CommandType numaralandırıcısı türünden 3 değer alabilir. Bu değerler ve ne işe yaradıkları aşağıdaki tabloda belirtilmiştir.

CommandType Değeri	Açıklaması
Text	Sql ifadelerini çalıştırmak için kullanılır. Bu aynı zamanda OleDbCommand sınıfına ait nesne örnekleri için varsayılan değerdir.
StoredProcedure	Veri kaynağında yer alan bir Saklı Yordam çalıştırılmak istendiğinde, CommandType değerine StoredProcedure verilir.
TableDirect	CommandType özelliğine bu değer atandığında, CommandText özelliği tablo adını alır. Komut çalıştırıldığında çalışan sql ifadesi "Select * From tabloadi" ifadesidir. Böylece belirtilen tablodaki tüm kayıtlar döndürülmüş olur.

Tablo 1. CommandType numaralandırıcısının değerleri.

Şimdi bu özelliklerin nasıl kullanılacağını tek tek incelemeye çalışalım. Öncelikle TableDirect değerinden başlayalım. Tek yapmamız gereken tüm satırlarını elde etmek istediğimiz tablo adını CommandText olarak belirtmek olacaktır. İşte örneğimiz.

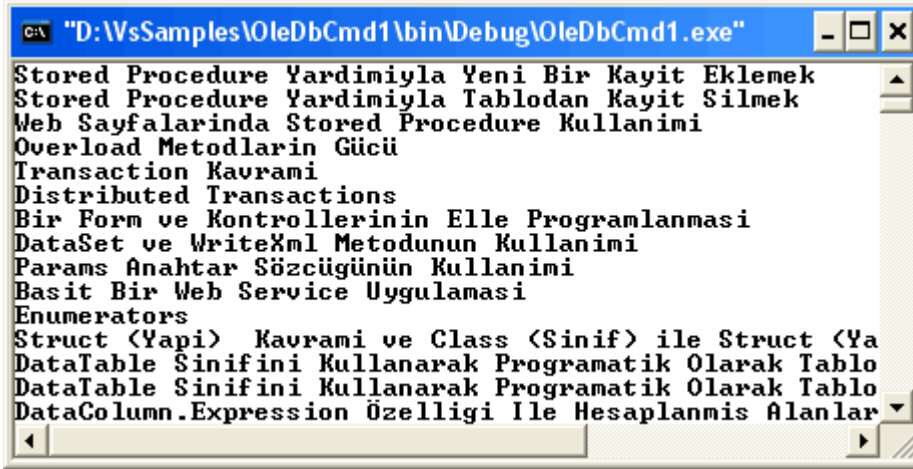
```
using System;
using System.Data.OleDb;
using System.Data;
namespace OleDbCmd1
{
    class Class1
```

```

{
    static void Main(string[] args)
    {
        OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated
security=sspi");
        OleDbCommand cmd=new OleDbCommand("Makale",con); /* Komut söz dizimi olarak tüm satırlarını almak istediğimi veri
tablosunun adını giriyoruz. */
        cmd.CommandType=CommandType.TableDirect; /* Komut tipimizi TableDirect olarak ayarlayıp, komut nesnemizin sql
ifadesinin Select * From Makale olmasını sağlıyoruz. */
        try
        {
            con.Open();
            /* Bir OleDbDataReader nesnesi tanımlayıp, komutumuzu ExecuteReader metodu ile çalıştırarak, sonuç kümesine ait
satırlardaki Konu alanının değerlerini ekrana yazdırıyoruz. */
            OleDbDataReader dr;
            dr=cmd.ExecuteReader();
            while(dr.Read())
            {
                Console.WriteLine(dr["Konu"].ToString());
            }
            dr.Close();
        }
        catch(Exception hata)
        {
            Console.WriteLine(hata.Message.ToString());
        }
        finally
        {
            con.Close();
        }
    }
}

```

Uygulamamızı çalıştırdığımızda, Makale tablosundaki tüm satırların alındığı sonuç kümesi içinden, Konu alanlarının değerlerinin ekrana yazdırıldığını görürüz.



Şekil 5. TableDirect sonucu.

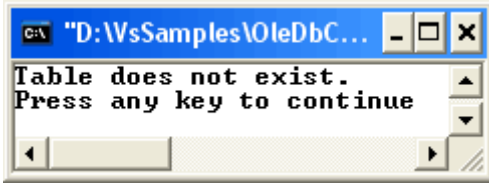
Ancak burada istisnai bir durum vardır. Bazı tablo isimleri içinde boşluklar olabilir. Örneğin "Site Adlari" isminde bir tablomuz olduğunu düşünelim. Böyle bir durumda TableDirect değerinin sonucunda bir istisnanın fırlatıldığını görürüz. Yukarıdaki örneğimizde tablo adı olarak Site Adlari verdiğimizizi düşünelim.

```
using System;
using System.Data.OleDb;
using System.Data;
namespace OleDbCmd1
{
    class Class1
    {
        static void Main(string[] args)
        {
            OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated
security=sspi");
            OleDbCommand cmd=new OleDbCommand("Site Adlari",con);
            cmd.CommandType=CommandType.TableDirect;
            try
            {
                con.Open();
                OleDbDataReader dr;
                dr=cmd.ExecuteReader();
                while(dr.Read())
                {
                    Console.WriteLine(dr["Baslik"].ToString());
                }
                dr.Close();
            }
        }
    }
}
```

```

    }
    catch(Exception hata)
    {
        Console.WriteLine(hata.Message.ToString());
    }
    finally
    {
        con.Close();
    }
}
}
}

```



Şekil 6. Tablonun olmadığı söyleniyor.

Bunun sebebi OleDbCommand nesnesinin tablo isminde yer alan boşlukları anlamamış olmasıdır. Bu nedenle aynı ifadeyi aşağıdaki şekilde değiştirmemiz gerekmektedir.

```
OleDbCommand cmd=new OleDbCommand("[Site Adlari]",con);
```

Bu durumda uygulamanın sorunsuz çalıştığını görürüz. OleDbCommand sınıfının CommandType özelliğinin diğer değeri ise 'StoredProcdeure' dür. Bu veri kaynağındaki saklı yordamlarının çağırılması için kullanılmaktadır. Bir saklı yordam kendisi için parametreler alabileceği gibi geriye değerlerde döndürebilir. Örneğin, Primary Key alanları üzerinden arama yapılan sorgularda Saklı Yordamların kullanılması son derece verimlidir. Nitekim kullanıcıların aramak için girdikleri her ID değeri için ayrı bir select sorgusu oluşturmak yerine, veri kaynağında bir nesne olarak yer alan ve ID değerini parametre olarak alan hazır, derlenmiş bir select ifadesini çalıştırmak daha verimli olacaktır. Örneğin Makale isimli tablomuzdan ID alanı 41 olan bir satırı elde etmek istiyoruz. Bu durumda, buradaki saklı yordamımıza bu ID değerini geçirmemiz ve dönen sonuçları almamız gerekiyor. Öncelikle saklı yordamımıza bir göz atalım.

```

ALTER PROCEDURE dbo.MakaleBul
(
    @MakaleID int
)
AS
SELECT * FROM Makale Where ID=@MakaleID
RETURN

```

Bu saklı yordam ID değerine @MakaleID isminde bir parametre alır. Bu parametre değeri ile tablodaki ilgili satır aranır. Bu satır bulunduğunda, geriye bu satırdaki tüm alanlar aktarılır. Şimdi uygulamamızda bunun nasıl kullanacağımızı görelim. Öncelikle OleDbCommand sınıfı nesne örneğimizi bu saklı yordam ismini CommandText değeri olacak şekilde oluştururuz. Daha sonra, CommandType özelliğine StoredProcedure değerini veririz. Böylece, CommandText özelliğindeki söz diziminin bir saklı yordamı temsil ettiğini ifade etmiş oluruz. Geriye parametre kalır. OleDbCommand sınıfı, parametrelerini OleDbParameterCollection koleksiyonunda birer OleDbParameter nesnesi olarak tutmaktadır. Dikkat etmemiz gereken nokta parametre adının , saklı yordamdaki ile aynı olmasıdır. Dilerseniz kodlarımızı yazalım ve bu işlemin nasıl yapıldığını görelim.

```
using System;
using System.Data.OleDb;
using System.Data;
namespace OleDbCmd1
{
    class Class1
    {
        static void Main(string[] args)
        {
            OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated security=sspi");
            OleDbCommand cmd=new OleDbCommand("MakaleBul",con); /* Çalıştırılacak sql ifadesi olarak saklı yordamımızın ismini giriyoruz. */
            cmd.CommandType=CommandType.StoredProcedure; /* CommandText ifadesinin, geçerli bağlantı nesnesinin temsil ettiği veri kaynağındaki bir saklı yordamı ifade ettiğini belirtiyor. */
            cmd.Parameters.Add("@MakaleID",OleDbType.Integer); /* Parametremiz oluşturuluyor. Adı @MakaleID, saklı yordamımızdaki ile aynı. Parametre tipi integer, nitekim Saklı Yordamımızdaki tipide int.*/
            cmd.Parameters["@MakaleID"].Value=41; /* Parametremizin değeri veriliyor. */
            try
            {
                con.Open();
                OleDbDataReader dr;
                dr=cmd.ExecuteReader();
                while(dr.Read())
                {
                    Console.WriteLine(dr["ID"].ToString()+"-"+dr["Konu"].ToString()+"-"+dr["Tarih"].ToString());
                }
                dr.Close();
            }
            catch(Exception hata)
            {
            }
```

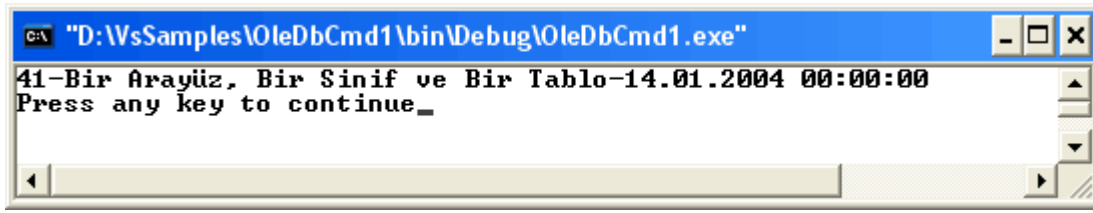


```

        Console.WriteLine(hata.Message.ToString());
    }
    finally
    {
        con.Close();
    }
}
}
}
}

```

Bu uygulamayı çalıştırdığımızda, aşağıdaki sonucu elde ederiz.



Şekil 7. Saklı Yordamın çalışmasının sonucu.

CommandType özelliğinin Text değeri varsayılandır. Text değeri, CommandText için yazılan sql ifadelerinin çalıştırılmasında kullanılır. Aslında bir saklı yordamı bu şekilde çağırabiliriz. Yani bir saklı yordamı, OleDbCommand sınıfının CommandType özelliğini Text olarak bırakarakta çağırabiliriz. Bunun için "{CALL MakaleBul(?)}" söz dizimini aşağıdaki örnekte olduğu gibi kullanırız. Sonuç aynı olacaktır. Burada, CALL ifadesinde parametrenin ? işareti ile temsil edildiğine dikkat edin. SqlCommand sınıfında bu parametreler @ParametreAdı olarak kullanılır.

```

OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated security=sspi");
OleDbCommand cmd=new OleDbCommand("{CALL MakaleBul(?)}",con);
cmd.Parameters.Add("@MakaleID",OleDbType.Integer);
cmd.Parameters["@MakaleID"].Value=41;
try
{
    con.Open();
    OleDbDataReader dr;
    dr=cmd.ExecuteReader();
    while(dr.Read())
    {
        Console.WriteLine(dr["ID"].ToString()+"-"+dr["Konu"].ToString()+"-"+dr["Tarih"].ToString());
    }
    dr.Close();
}

```

```

}
catch(Exception hata)
{
    Console.WriteLine(hata.Message.ToString());
}
finally
{
    con.Close();
}

```

Gelelim, OleDbCommand sınıfının diğer önemli üyelerine. Bu üyelerden birisi CommandTimeout özelliğidir. Bir sql ifadesi , OleDbCommand nesnesi tarafından çalıştırıldığında, ilk sonuçlar döndürülene kadar belli bir süre geçer. İşte bu sürenin uzunluğu CommandTimeout özelliği tarafından belirlenir. Başlangıç değeri olarak 30 saniyeye ayarlanmıştır. Bu süre zarfında komut çalıştırılması ile birlikte herhangi bir sonuç döndürülemez ise, bir istisna fırlatılır. Bu özellik aslında, ilk sonuçların dönmeye başlaması için ne kadar süre bekleneceğini belirtir. Düşününkü, bir OleDbAdapter nesnesinin çalıştırdığı bir OleDbCommand nesnemiz var. Bu komutun işaret ettiği sql ifadesi çalıştırıldıktan sonra, CommandTimeout'ta belirtilen sürede bir sonuç dönmez ise, süre aşımı nedeni ile bir istisna oluşur. Ancak burada özel bir durum vardır. Eğer, bu süre içinde belli bir sayıda kayıt(en azından tek satırlık veri) geri dönmüş ise, CommandTimeout süresi geçersiz hale gelir. Böyle bir durumda OleDbDataAdapter tarafından döndürülmeye çalışılan kayıtların hepsi elde edilene kadar komut çalışmaya devam eder. Bu bazen bir ömür boyu sürebilir. Buda tam anlamıyla bir ironidir.

Daha önceki makalelerimizde hatırlayacağınız gibi, iş parçacıklarının çok katlı mimaride önemli bir yeri vardır. Bir OleDbCommand sınıfı nesne örneğini bir iş parçacığı olarak çalıştırmak için, Transaction özelliğine, iş parçacığını üstlenen OleDbTransaction nesnesini atamamız gerekir. Bununla ilgili basit bir örnek aşağıda verilmiştir.

```

using System;
using System.Data.OleDb;
using System.Data;
namespace OleDbCmd1
{
    class Class1
    {
        static void Main(string[] args)
        {
            OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated security=sspi");
            OleDbCommand cmd=new OleDbCommand("Insert Into [Site Adlari] (Baslik,Adres,Resim,Icerik)
            Values('C#','www.csharpnedir.com','images/resim1.jpg','C# üzerine her türlü makale.')" ,con);
            con.Open();
            OleDbTransaction trans=con.BeginTransaction(); /* Transaction'ımız, geçerli bağlantımızı için yaratılıyor. */

```

```

cmd.Transaction=trans; /* Komutumuzun tanımlanan bağlantı için açılmış transaction içinde bir iş parçacığı olarak çalışacağı
belirleniyor. */
int sonuc=cmd.ExecuteNonQuery();
if(sonuc==1)
{
trans.Commit(); /* Komut başarılı bir şekilde çalıştırılmışsa Commit ile tüm işlemler onaylanıyor. */
Console.WriteLine(sonuc.ToString()+" Kayıt Girildi...");
}
else
{
trans.Rollback(); /* Komut başarısız ise tüm işlemler geri alınıyor. */
}
}
}
}

```

OleDbCommand sınıfının kullanıldığı diğer önemli bir yerde OleDbDataAdapter sınıfıdır. Nitekim bu sınıfın içerdiği SelectCommand, UpdateCommand, DeleteCommand, InsertCommand özellikleri, OleDbCommand sınıfı türünden nesneleri değer olarak alırlar. Bu konuyu ilerleyen makalelerimizde OleDbDataAdapter sınıfını işlerken incelemeye çalışacağız.

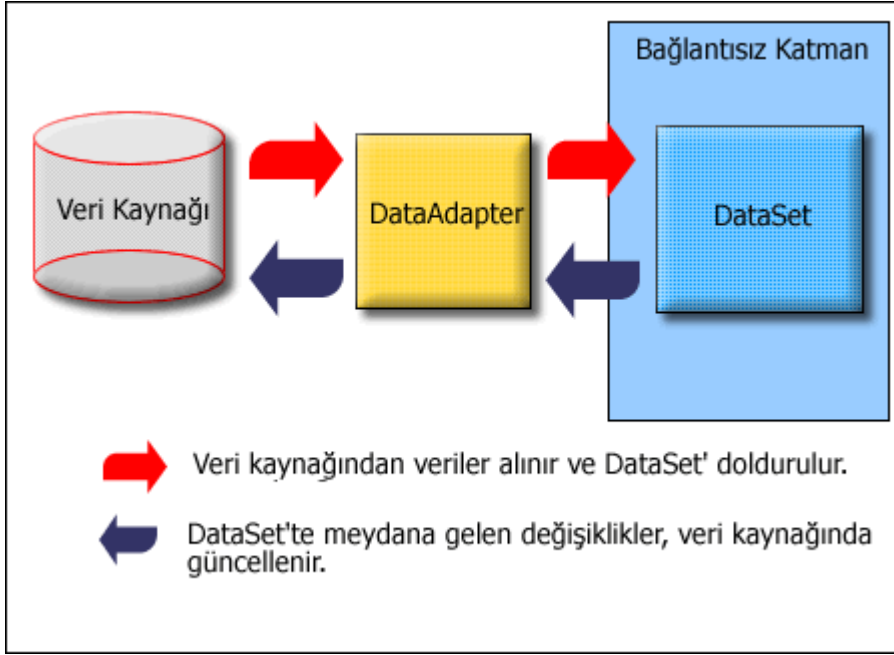
Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

DataAdapter Kavramı ve OleDbDataAdapter Sınıfına Giriş

Değerli Okurlarım, Merhabalar.

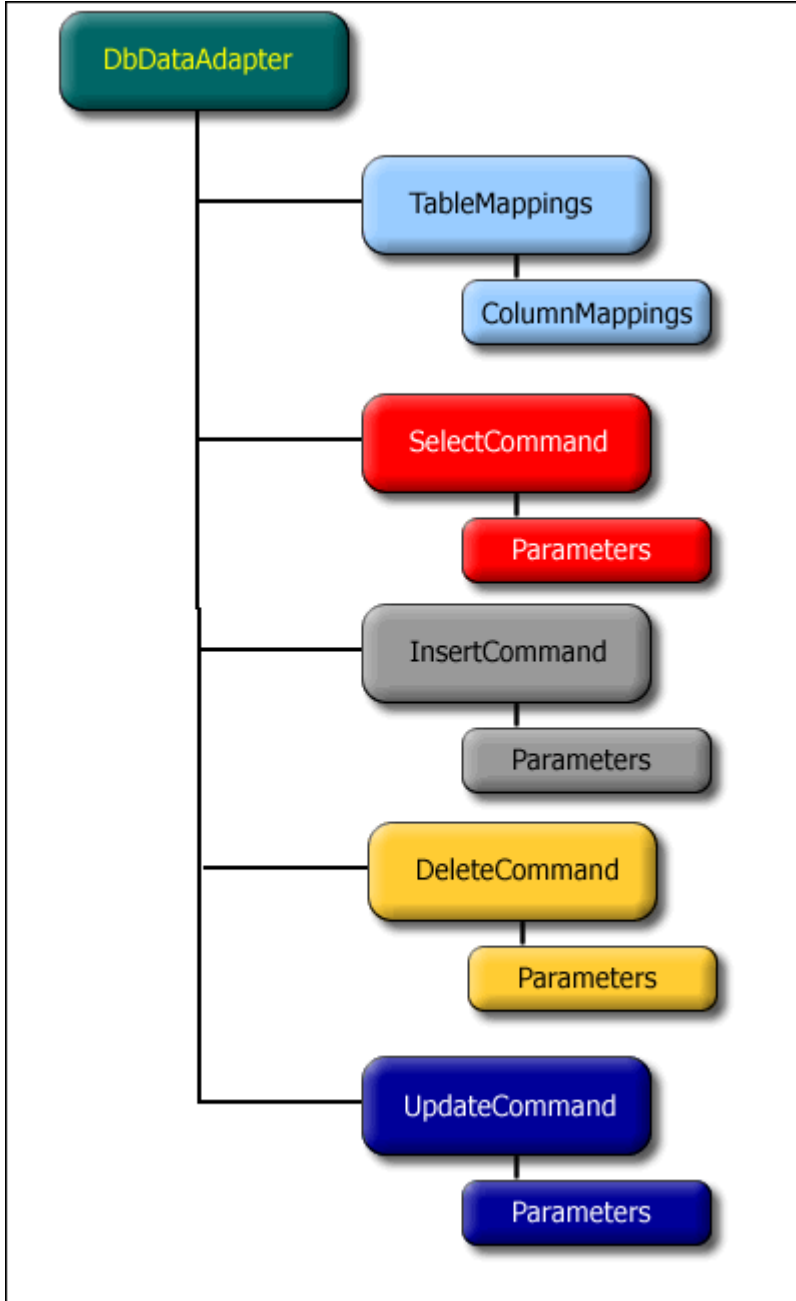
Bu makalemizde, Ado.Net'in en çok kullanılan kavramlarından birisi olan DataAdapter kavramını incelemeye çalışacak ve OleDbDataAdapter sınıfına kısa bir giriş yapacağız. Pek çok programcı, veritabanı uygulamaları geliştirirken, kontrol ve performansa büyük önem verir. Ancak aynı zamanda bu kazanımlara kolay yollar ile ulaşmak isterler. Ado.Net modelinde, bağlantısız katman ile bağlantılı katman arasındaki iletişim ve veri alışverişinin, kontrol edilebilir, performanslı ve aynı zamanda kolay geliştirilir olmasında DataAdapter kavramının yeri yadırganamayacak kadar fazladır. Temel tanım olarak, DataAdapter sınıfları, sahip oldukları veri sağlayıcılarının izin verdiği veri kaynakları ile, sistem belleği üzerinde yer alan bağlantısız katman nesneleri arasındaki veri alışverişinin kolay, güçlü ve verimli bir şekilde sağlanmasından sorumludurlar. Bu tanımdan yola çıkarak, DataAdapter sınıflarının, veri kaynağından verilerin alınıp, bağlantısız katman nesneleri olan DataSet ve DataTable nesnelerine doldurulmasından sorumlu olduğunu; ayrıca, bağlantısız katman nesnelerinin taşıdığı verilerdeki değişikliklerinde veri kaynağına yansıtılmasından sorumlu olduğunu söyleyebiliriz. İşte bu, DataAdapter sınıfının rolünü tam olarak açıklayan bir tanımlamadır.

Veritabanı uygulamalarında en önemli unsurların, verinin taşınması, izlenebilmesi, üzerinde değişikliklerin yapılması ve tekrar veri kaynağına yansıtılması olduğunu söyleyebiliriz. DataAdapter sınıflarının bu unsurların gerçekleştirilmesinde çok önemli rol oynadıkları bir gerçektir. Aşağıdaki şekil DataAdapter sınıflarının işleyişini daha iyi anlamamıza yardımcı olacaktır.



Şekil 1. DataAdapter Sınıfının Rolü.

Ado.net modeli, .net Framework'ün en son sürümünde, OleDb veri kaynakları için OleDbDataAdapter, Sql veri kaynakları için SqlDataAdapter, Oracle veri kaynakları için OracleDataAdapter ve Odbc veri kaynakları içinde OdbcDataAdapter sınıflarına sahiptir. DataAdapter sınıflarının yapısı aşağıdaki gibidir.



Şekil 2. DataAdapter sınıflarının genel yapısı.

Şimdi dilerseniz bu üyelerin hangi amaçlar için kullanıldığına kısaca değinelim. Her DataAdapter sınıfı mutlaka bir SelectCommand sorgusu içermek zorundadır. Nitekim, bir DataAdapter nesnesinin yaratılış amacı, veri kaynağından çekilen verileri bağlantısız katmandaki ilgili nesnelere aktarmaktır. İşte bunun için, SelectCommand özelliği geçerli bir sorguya sahip olmalıdır. Veri almak için kullanılan sorgu bir Command nesnesine işaret etmektedir. Çoğu zaman bir DataAdapter oluştururken bu sınıfın kurucusu içinde, select sorgularını string olarak gireriz. Bu string aslında sonradan bir Command nesnesi haline gelerek, DataAdapter sınıfının SelectCommand özelliğine atanır.

Diğer yandan, verileri yüklü olan bir DataSet içerisinde yapılan değişikliklerin veri kaynağına tekrardan aktarılması için, yapılan değişikliğin çeşidine uygun komutların DataAdapter tarafından oluşturulması gerekmektedir. Örneğin, DataSet üzerinde yeni satırlar girilmiş ise, bunlar InsertCommand özelliğinin sahip olduğu sql metni ile veri kaynağına aktarılır. Aynı şekilde, satır silme işlemleri için DeleteCommand özelliğindeki sql söz dizimi, güncelleme işlemleri için ise UpdateCommand özelliğindeki sql ifadeleri kullanılır. Bu özelliklerin tümünü elle oluşturabileceğiniz gibi, Visual Studio.Net ortamında yada CommandBuilder sınıfını vasıtasıyla otomatik olarak oluşturulmasını sağlayabilirsiniz. Bu özellikleri sonraki maddelerimizde incelemeye çalışacağız.

TableMappings DataTableMappingCollection türünden bir koleksiyondur. Görevi ise oldukça ilginçtir. Bir DataAdapter nesnesi ile veri kaynağındaki bir tabloyu, DataSet' aktardığımızda, aktarma ve güncelleme işlemleri sırasında alan isimlerinin her iki katmandada eşleştirilebilmesi için kullanılır. Nitekim, DataAdapter ile bir tabloyu elde ettiğimizde bu tablo, Table ismi ile işleme sokulur. Çünkü OleDbDataAdapter tablonun ismi ile ilgilenmez. Fakat biz dilersek TableMappings koleksiyonuna bir DataTableMapping nesnesi ekleyerek tablo adının daha uygun bir isme işaret etmesini sağlayabiliriz. Aynı konu, sütun adları içinde geçerlidir. Bazen tablodaki sütun adları istediğimiz isimlerde olmayabilir. İşte bu durumda, DataTableMapping nesnesinin ColumnMappings koleksiyonu kullanılır. DataSet içindeki her tablo DataColumnMapping türünden nesneler içeren bir ColumnMappings koleksiyonuna sahiptir. Bu nesnelerin her biri veri kaynağındaki tabloda yer alan sütun adlarının, DataSet içindeki tabloda hangi isimlere karşılık geldiğinin belirlenmesinde kullanılır. Bu konu şu an için karışık görünmüş olabilir ancak ilerleyen örneklerimizde bu konuya tekrar değinecek, daha anlaşılır olması için çalışacağız.

DataAdapter sınıflarının genel özelliklerine değindikten sonra derseniz OleDbDataAdapter sınıfımızı incelemeye başlayalım. OleDbDataAdapter sınıfı System.Data.OleDb isim uzayı içinde yer almaktadır. Bir OleDbDataAdapter nesnesi yaratmak için kullanabileceğimiz 4 adet aşırı yüklenmiş yapıcı metod bulunmaktadır. Bunlar aşağıdaki tabloda belirtilmiştir.

Yapıcı Metod Prototipi	Açıklaması
public OleDbDataAdapter(string, string);	Select sorgusunu ve bağlantı için gerekli söz dizimini metin şeklinde parametre olarak alır.
public OleDbDataAdapter(string, OleDbConnection);	Sorguyu metin bazında alırken, bağlantı için önceden oluşturulmuş bir OleDbConnection nesnesini parametre olarak alır.
public OleDbDataAdapter(OleDbCommand);	Select sorgusunu ve geçerli bir bağlantıyı işaret eden bir OleDbCommand nesnesini parametre olarak alır.
public OleDbDataAdapter();	Böyle oluşturulan bir OleDbDataAdapter'ı kullanabilmek için, ilgili özellikler (SelectCommand gibi.) sonradan ayarlanır.

Tablo 1. OleDbDataAdapter sınıfının yapıcı metodları.

Şimdi burada önemli olan bir iki noktayı vurgulamamız gerekiyor. Herşeyden önce bir OleDbDataAdapter nesnesinin kullanılma amacı, bağlantısız katman nesnesini veri kaynağındaki veriler ile doldurmaktır. Bu her DataAdapter sınıfı içinde öncelikli hedeftir. Bu nedenle her DataAdapter nesnesinin mutlaka sahip olması gereken bir select sorgu ifadesi dolayısıyla var olan bir SelectCommand özelliğine ihtiyacı vardır. Ayrıca diğer önemli gereklilik, geçerli bir bağlantının yani bir Connection nesnesinin olmasıdır. Şimdi dilerseniz, örnekler ile OleDbDataAdapter nesnelerinin nasıl oluşturulduğunu ve verilerin, veri kaynağından bağlantısız katman nesnelere nasıl çekildiğini görelim.

İlk örneğimizde, bir windows uygulamasındaki DataGrid nesnemizi, bağlantısız katmanda çalışacak DataSet nesnemize bağliayacağız. DataSet nesnemizi veriler ile doldurmak için, OleDbDataAdapter nesnemizi kullanacağız. İşte uygulamamızın kısa kodları.

```
private void btnDoldur_Click(object sender, System.EventArgs e)
{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");
    string sqlIfadesi="Select * From Makale";
    OleDbDataAdapter daFriends=new OleDbDataAdapter(sqlIfadesi,conFriends);
    DataSet ds=new DataSet();
    daFriends.Fill(ds);
    dgMakale.DataSource=ds;
}
```

Kodlarımızı inceleyecek olursak; öncelikle OleDbDataAdapter nesnemiz için gerekli ve olmasa olmaz üyeleri tanımladık. Öncelikle geçerli bir bağlantı hattımızın olması gerekiyor. Bu amaçla bir OleDbConnection nesnesi kullanıyoruz. Diğer yandan, OleDbDataAdapter nesnesinin oluşturduğumuz DataSet nesnesini doldurabilmesi için, veri kaynağından veri çekebileceği bir sql ifadesine ihtiyacımız var. Bu amaçlada bir sql cümleciğini string olarak oluşturuyoruz. Sonraki adımda ise OleDbDataAdapter nesnemizi yaratıyoruz. Burada kullanılan yapıcı metod, sql ifadesini string olarak alıyor ve birde bağlantı hattını temsil edecek olan OleDbConnection nesnesini parametre olarak alıyor. Daha sonra, veri kaynağından bu sql ifadesi ile çekilecek verilerin bellekte tutulduğu bölgeyi referans edecek DataSet nesnemiz oluşturuluyor. Burada Fill metodu, sql ifadesini, geçerli bağlantı hattı üzerinde çalıştırıyor ve elde edilen veri kümesini, metod parametresi olarak aldığı DataSet nesnesinin bellekte gösterdiği adrese yerleştiriyor. Uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 3. Fill Metodu ile DataSet'in doldurulması.

OleDbDataAdapter nesnesinin Fill metodunu çağırdığımızda, nesne, parametre olarak aldığı bağlantıyı otomatik olarak açmaktadır. Yani Fill metodunu çağırmadan önce bağlantı nesnemizi Open metodu ile açmamıza gerek yoktur. Fill metodu ile DataSet nesnesi doldurulduktan sonra ise, OleDbDataAdapter, açık olan bağlantıyı otomatik olarak kapatacaktır. Bu kolaylık birden fazla tabloyu bir DataSet içerisine farklı OleDbDataAdapter nesneleri ile alacağımız zaman dezavantajdır. Nitekim her bir OleDbDataAdapter nesnesi eğer aynı bağlantıyı kullanıyorlarsa, her defasında veri kaynağına olan bağlantıyı açacak ve kapatacaktırlar. Söz gelimi aşağıdaki kodları göz önüne alalım.

```
private void btnDoldur_Click(object sender, System.EventArgs e)
{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");

    string sqlIfadesi1="Select * From Makale";
    string sqlIfadesi2="Select * From Kitap";

    OleDbDataAdapter daMakale=new OleDbDataAdapter(sqlIfadesi1,conFriends);
    OleDbDataAdapter daKitap=new OleDbDataAdapter(sqlIfadesi2,conFriends);

    DataSet ds=new DataSet();
    daMakale.Fill(ds);
    daKitap.Fill(ds);
    dgMakale.DataSource=ds;
}
```


Bu uygulamada iki OleDbDataAdapter nesnesi aynı bağlantıyı kullanarak farklı veri tablolarını aynı DataSet içerisine yüklemiştir. Her bir Fill metodu çağırıldığında bağlantı açılır, tablodan veriler, sql ifadeleri gereği alınır, DataSet nesnesinin referans ettiği bellek bölgesine yüklenir ve açık olan bağlantı kapatılır. Bu durumda, uygulamamızdaki kodlarda bağlantının iki kere açılıp kapatıldığını söyleyebiliriz. Bu gözle görülür bir performans kaybına yol açmıyabilir ancak programlama tekniği açısından bağlantının bu kadar kısa süreler için tekrardan açılıp kapatılması sistem kaynaklarını boş yere kullanmak manasınada gelmektedir. Peki ne yapılabilir? OleDbDataAdapter sınıfının bir özelliği, eğer kullanılan bağlantı açık ise, OleDbDataAdapter bu bağlantıyı biz kapatana kadar kapatmayacak olmasıdır. Yani yukarıdaki kodu şu şekilde düzenlersek istediğimiz sonuca ulaşabiliriz.

```
private void btnDoldur_Click(object sender, System.EventArgs e)
{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");

    string sqlIfadesi1="Select * From Makale";
    string sqlIfadesi2="Select * From Kitap";

    OleDbDataAdapter daMakale=new OleDbDataAdapter(sqlIfadesi1,conFriends);
    OleDbDataAdapter daKitap=new OleDbDataAdapter(sqlIfadesi2,conFriends);

    DataSet ds=new DataSet();

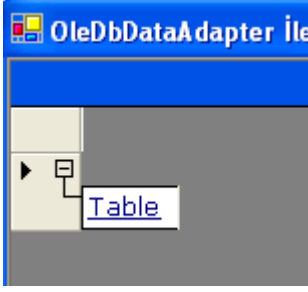
    conFriends.Open();

    daMakale.Fill(ds);
    daKitap.Fill(ds);

    dgMakale.DataSource=ds;

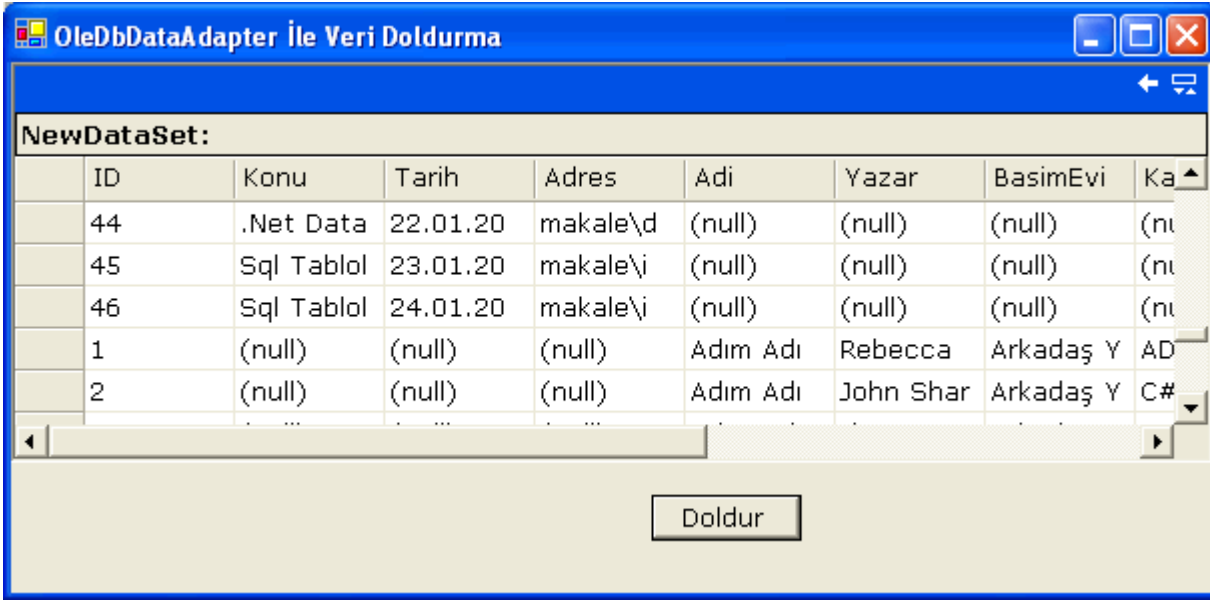
    conFriends.Close();
}
```

Bu durumda, Fill metodları bağlantı durumunu gözleyecek ve eğer bağlantı açık ise herhangi bir müdahalede bulunmadan bu açık bağlantı üzerinden işlemleri gerçekleştirecektir. Fill metodu işi bittiğinde, kullandığı bu açık bağlantıyı kapatmayacaktır. Bu sayede ardından gelen ve aynı bağlantıyı kullanan OleDbDataAdapter nesnesi, yeni bir bağlantı açmaya gerek duymayacak, halen açık olan bağlantıyı kullanacaktır. Burada unutulmaması gereken nokta, bağlantı nesnemiz ile işlem bittiğinde bu nesneyi Close metodu ile kapatmaktır. Şimdi gelelim Fill metodundaki başka bir noktaya. Yukarıdaki son örneğimizi çalıştıracak olursak aşağıdaki görüntüyü elde ederiz.



Şekil 4. İki tabloda aynı isim altında yüklenir.

Görüldüğü gibi iki tablomuzda DataSet'e tek bir tablo ismi altında yüklenmiştir. Bu isme tıkladığımızda, ilk önce Makale verilerinin görüldüğünü ardından Kitap tablosuna ait verilerin görüldüğünü anlarız.



Şekil 5. Her iki tabloya ait verilerin görünümü.

Bu elbette istemediğimiz bir durumdur. Bunu düzeltmek için, Fill metodunun aşağıdaki prototipi verilen aşırı yüklenmiş halini kullanırız.

```
public int Fill(DataSet dataSet,string srcTable);
```

Burada ikinci parametre aktarılan tablo için bir ismi string olarak almaktadır. Böylece, DataSet içerisine aktarılan tabloları isimlendirebiliriz. Nitekim OleDbDataAdapter sınıfı, Fill metodu ile tablolardaki verileri DataSet içine alırken, sadece alan adlarını eşleştirmek için alır. Tablo adları ile ilgilenmez. Bu nedenle bir tablo ismi belirtmesek, bu DataSet içerisine Table ismi ile alınacaktır. Biz Fill metoduna bir tablo ismini parametre olarak verdiğimizde, DataAdapter sınıfının TableMappings koleksiyonu, DataSet içinde bizim verdiğimiz tablo ismini, veri kaynağındaki ile eşleştirir. Dolayısıyla yukarıdaki kodları aşağıdaki gibi düzenlersek sonuç istediğimiz gibi olacaktır.

```
private void btnDoldur_Click(object sender, System.EventArgs e)
```

```

{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");

    string sqlIfadesi1="Select * From Makale";
    string sqlIfadesi2="Select * From Kitap";

    OleDbDataAdapter daMakale=new OleDbDataAdapter(sqlIfadesi1,conFriends);
    OleDbDataAdapter daKitap=new OleDbDataAdapter(sqlIfadesi2,conFriends);

    DataSet ds=new DataSet();

    conFriends.Open();

    daMakale.Fill(ds,"Makaleler");
    daKitap.Fill(ds,"Kitaplar");

    dgMakale.DataSource=ds;

    conFriends.Close();
}

```

Uygulamamızı çalıştırdığımızda aşağıdaki sonucu alırız.



Şekil 6. Fill metodunda Tablo isimlerinin verilmesi.

Bazı durumlarda, toplu sorgular (batch queries) çalıştırmak isteyebiliriz. Örneğin aşağıdaki kodları ele alalım. Burada, 3 sorgunun yer aldığı bir toplu sorgu cümlecisi yer almaktadır. OleDbDataAdapter nesnemiz için, bu sql cümlecisini kullandığımızda sonuç kümelerinin Table, Table1 ve Table2 isimleri ile DataSet içerisine alındığını görürüz.

```

private void btnDoldur_Click(object sender, System.EventArgs e)
{

```

```

OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");
string sqlIfadesi="Select * From Makale;Select * From Kitap;Select * From Kisiler";
OleDbDataAdapter daMakale=new OleDbDataAdapter(sqlIfadesi,conFriends);
DataSet ds=new DataSet();
daMakale.Fill(ds);
dgMakale.DataSource=ds;
}

```



Şekil 7. Toplu Sorguların Çalıştırılmasının Sonucu.

Bu elbette uygulamamızın görselliği açısından çok hoş bir durum değildir. Yapabileceğimiz işlem ise, OleDbDataAdapter nesnemizin TableMappings koleksiyonuna, tabloların görmek istediğimiz asıl isimlerini eklemek olacaktır. Bu amaçla yazmış olduğumuz kodları aşağıdaki gibi değiştirmeliyiz.

```

private void btnDoldur_Click(object sender, System.EventArgs e)
{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");
    string sqlIfadesi="Select * From Makale;Select * From Kitap;Select * From Kisiler";
    OleDbDataAdapter da=new OleDbDataAdapter(sqlIfadesi,conFriends);

    da.TableMappings.Add("Table","Makaleler");
    da.TableMappings.Add("Table1","Kitaplar");
    da.TableMappings.Add("Table2","Arkadaslarim");

    DataSet ds=new DataSet();
    da.Fill(ds);
    dgMakale.DataSource=ds;
}

```

Burada yapılan işlemi açıklayalım. OleDbDataAdapter nesnemizin, TableMappings koleksiyonu, veri kaynağındaki tablo isimlerinin, uygulama içerisindeki bağlantısız katman nesnesi içerisinde nasıl isimlendirileceğini belirtmektedir. Dolayısıyla Fill metodunu

çağırdığımızda ilk sorgu sonucu elde edilen Table isimli tablo, Makaleler olarak, ikinci sorgu sonucu elde edilen sonuç kümesini temsil edilen Table1 tablosu, Kitaplar olarak ve sonundada Table2 ismiyle gelen son tablo Arkadaslarım olarak DataSet içerisine alınacaktır. TableMappings burada, sadece isimlerin eşleştirilmesinde rol oynar. Aynı şekilde, OleDbDataAdapter nesnemize ait Update metodunu kullandığımızda, TableMappings koleksiyonunda bu tablo isimlerini birbirleri ile eşleştirilerek, doğru tabloların doğru kaynaklara yönlendirilmesi sağlanmış olur.



Şekil 8. TableMappings Koleksiyonunun Önemi.

Şu ana kadarki örneklerimizde, bağlantısız katman nesnesi olarak DataSet'i kullandık. DataSet birden fazla veri tablosunu DataTable nesneleri olarak bünyesinde barındıran kuvvetli bir sınıftır. Ancak çoğu zaman uygulamalarımızda sadece tek tabloyu bağlantısız katmanda kullanmak isteyebiliriz. Böyle bir durumda bu tek tablo verisi için, DataSet nesnesi kullanmak sistem kaynaklarını daha çok harcamak anlamına gelir. Bunu çözmek için, veri kaynağından okunan verileri bir DataTable nesnesine aktarırız. İşte OleDbDataAdapter nesnesinin Fill metodu ile DataTable nesnelerini de doldurabiliriz. Bunun için aşağıdaki kodlarda belirtilen tekniği uyguluyoruz.

```
private void btnDoldur_Click(object sender, System.EventArgs e)
{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");
    string sqlIfadesi="Select * From Makale";
    OleDbDataAdapter da=new OleDbDataAdapter(sqlIfadesi,conFriends);
    DataTable dt=new DataTable("Makalelerim");
    da.Fill(dt);
    dgMakale.DataSource=dt;
    dgMakale.CaptionText=dt.TableName.ToString();
}
```

Burada DataTable nesnemizi oluştururken parametre olarak String bir değer girdiğimize dikkat edelim. Bu değer, verilerin alındığı kümenin, hangi isimde bir tabloya işaret edeceğini belirtmektedir. Fill metodunun kullanım şeklinde ise parametre olarak DataTable nesnesini alan aşağıdaki prototip kullanılmıştır.

```
public int Fill (DataTable dataTable);
```

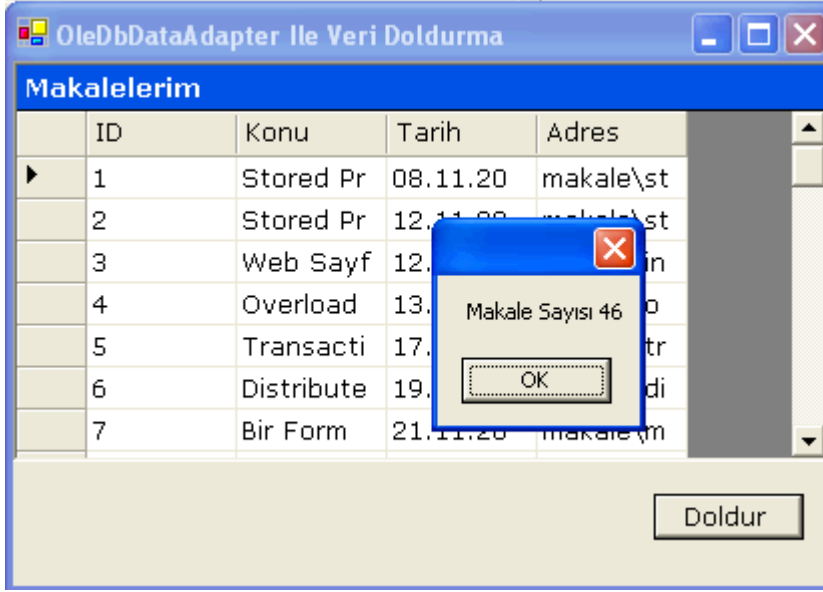
Son kodlarımızı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 9. Fill metodu ile verilerin DataTable'a aktarılması.

Fill metodunda dikkati çeken bir diğer nokta, döndürdüğü integer tipteki değerdir. Bu dönüş değeri, OleDbDataAdapter nesnesinin çalıştırdığı sorgu sonucu dönen satır sayısına işaret etmektedir. Söz gelimi yukarıdaki örneğimizi ele alalım. OleDbDataAdapter nesnemizin dönüş değerini kontrol ettiğimizde, tablomuzdan okunan satır sayısının döndürüldüğünü görmüş oluruz.

```
private void btnDoldur_Click(object sender, System.EventArgs e)
{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");
    string sqlIfadesi="Select * From Makale";
    OleDbDataAdapter da=new OleDbDataAdapter(sqlIfadesi,conFriends);
    DataTable dt=new DataTable("Makalelerim");
    int SatirSayisi=da.Fill(dt);
    dgMakale.DataSource=dt;
    dgMakale.CaptionText=dt.TableName.ToString();
    MessageBox.Show("Makale Sayısı "+SatirSayisi.ToString());
}
```



Şekil 10. Fill metodundan dönen değer ile kayıt sayısının öğrenilmesi.

Fill metodunun aşağıda prototipi verilen aşırı yüklenmiş halini kullanarak, belli bir satırdan itibaren belirli bir sayıda kaydın elde edilmesini sağlayabiliriz.

```
public int Fill(DataSet, int, int, string);
```

Burada Fill metodu dört parametre almaktadır. İlk parametremiz verilerin ekleneceği DataSet nesnesi ve son parametrede eklenecek verilerin temsil edileceği tablo adıdır. İkinci ve üçüncü parametreler integer tipte değerler alırlar. İkinci parametre sorgu sonucu elde edilen kayıt kümesinin hangi satırdan itibaren okuma yapılacağını, üçüncü parametre ise kaç satır alınacağını belirtmektedir. Örneğin Makale isimli tablomuzdaki verileri, tarih sırasına göre tersten çeken bir sql sorgumuz olduğunu düşünelim. İlk 3 satırı elde edip DataSet içindeki ayrı bir tabloya almak istediğimizi varsayalım. Böylece tabloya eklenen son 3 Makaleye ait bilgilere erişmiş olacağız. Bunun için aşağıdaki tekniği kullanacağız.

```
private void btnDoldur_Click(object sender, System.EventArgs e)
{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");
    string sqlIfadesi="Select * From Makale Order By Tarih Desc";
    OleDbDataAdapter da=new OleDbDataAdapter(sqlIfadesi,conFriends);
    DataSet ds=new DataSet("Makaleler");
    int SatirSayisi=da.Fill(ds,0,3,"Son3Makale");
    dgMakale.DataSource=ds;
}
```

Burada Fill metodunda select sorgusu sonucu elde edilen kümede 0 indisli satırdan (yani ilk satır) itibaren 3 satır verinin okunmasını ve Son3Makale isimli tabloya aktarılmasını sağlıyoruz. İşte sonuç.



Şekil 11. Fill metodu ile belli bir satırdan itibaren belli sayıda satır almak.

Fill metodu ile, veri kaynağındaki verileri bağlantısız katmana çekerken saklı yordamları kullanmak isteyebiliriz. Bu amaçla, OleDbDataAdapter sınıfının SelectCommand özelliğine, bu saklı yordamı çalıştırmak için kullanılacak bir OleDbCommand nesnesini atamak kullanabileceğimiz tekniklerden birisidir. OleDbCommand nesnesini yaratırken kullandığımız new yapılandırıcısına ait aşırı yüklenmiş hallerden birisi aşağıdaki gibiydi.

```
public OleDbDataAdapter(OleDbCommand);
```

Burada OleDbCommand nesnesini, saklı yordamımızı çalıştıracak şekilde oluştururuz. Aşağıdaki örnek saklı yordamımızı göz önüne alalım.

```
CREATE PROCEDURE Makaleler
AS
Select * From Makale
RETURN
```

Şimdi bu saklı yordamımızı çalıştıracak OleDbDataAdapter nesnemiz için gerekli kodlamaları yapalım.

```
private void btnDoldur_Click(object sender, System.EventArgs e)
{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");
    OleDbCommand cmd=new OleDbCommand("Makaleler",conFriends);
    cmd.CommandType=CommandType.StoredProcedure;
    OleDbDataAdapter da=new OleDbDataAdapter(cmd);
    DataSet ds=new DataSet();
```



```

da.Fill(ds,"TumMakaleler");
dgMakale.DataSource=ds;
}

```

Burada OleDbDataAdapter nesnemizi, OleDbCommand nesnemizi kullanacak şekilde oluşturduk. Dolayısıyla, OleDbDataAdapter, OleDbCommand nesnesinin belirttiği sql ifadesini, yine OleDbCommand nesnesinin kullandığı bağlantı üzerinden çalıştırmaktadır. OleDbCommand nesnemiz bir saklı yordama işaret ettiği için, OleDbDataAdapter sonuç olarak bu saklı yordamı çalıştırmış olur. Böylece DataSet nesnemizin bellekte işaret ettiği bölge, saklı yordamın çalışması sonucu dönen veriler ile doldurulmuş olacaktır.



Şekil 12. OleDbDataAdapter ile Saklı yordamın çalıştırılması.

Tabi bu amaçla illede bir OleDbCommand nesnemiz kullanmak şart değildir. Aynı işlem için aşağıdaki söz dizimini sql ifadesi olarak SelectCommand özelliği için belirleyebiliriz.

```
{Call Makaleler}
```

Bu durumda kodlarımızı aşağıdaki gibi değiştirmemiz gerekmektedir.

```

private void btnDoldur_Click(object sender, System.EventArgs e)
{
    OleDbConnection conFriends=new OleDbConnection("Provider=SQLOLEDB;Data Source=localhost;initial
catalog=Friends;integrated security=SSPI");
    OleDbDataAdapter da=new OleDbDataAdapter("{CALL Makaleler}",conFriends);
    DataSet ds=new DataSet();
    da.Fill(ds,"TumMakaleler");
    dgMakale.DataSource=ds;
}

```

Burada dikkat edilmesi gereken bir nokta vardır. SqlDataAdapter nesneleri için bu çağırım {EXEC Makaleler} şeklindedir. Buraya kadar anlattıklarımızla OleDbDataAdapter sınıfı ile ilgili olarak bayağı bir yol katettiğimizi düşünüyorum. Bir sonraki makalemizde, OleDbDataAdapter sınıfını incelemeye devam edeceğiz. Öncelikle OleDbDataAdapter nesnelerinin Visual Studio.Net ortamında kolayca nasıl oluşturulduklarını inceleyeceğiz. Böylece OleDbDataAdapter sınıfı için gereken SelectCommand, InsertCommand, DeleteCommand, UpdateCommand özelliklerinin nasıl otomatik olarak oluşturulduğunu anlayacağız. Daha sonra aynı iş için CommandBuilder sınıfının nasıl kullanıldığını inceleyeceğiz. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

OleDbDataAdapter Sınıfı - 2 (Birincil Anahtarların Önemi ve OleDbDataAdapter Nesnesini Otomatik Oluşturmak)

Değerli Okurlarım, Merhabalar.

Önceki makalemizde, OleDbDataAdapter sınıfının ne işe yaradığından bahsetmiş ve kısa bir giriş yapmıştık. Bu makalemizde, OleDbDataAdapter sınıfının diğer önemli unsurlarını incelemeye devam edeceğiz. İncelemek istediğim ilk konu, OleDbDataAdapter nesnesi yardımıyla, ilişkisel veritabanı modellerinden bağlantısız katmana aktarılan tabloların, sahip olduğu birincil anahtar (Primary Key) ve kısıtlamaların (Constraints) ne şekilde irdelendiği olacak. Konuyu iyice kavrayabilmek amacıyla aşağıdaki basit örnek ile incelememize başlayalım. Bu örneğimizde, sql sunucumuzda yer alan bir tabloya ait verileri DataSet üzerine alıyor ve alınan alanların bir takım bilgilerini okuyoruz. Örneğin, alanların veri tipi, boyutu, null değerler içerip içermediği ve alan adları bilgilerini ekrana yazdırıyoruz.

```
using System;
using System.Data;
using System.Data.OleDb;
```

```
namespace OleDbDA2
```

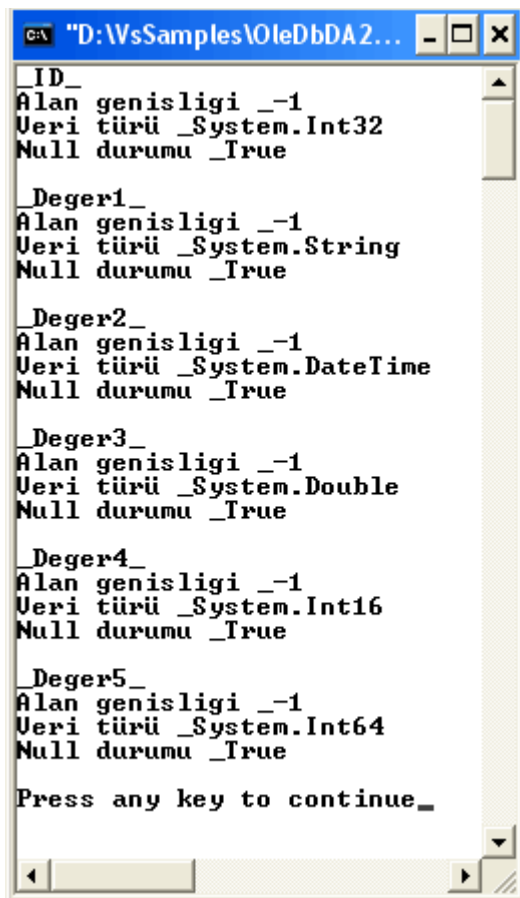
```
{
    class Class1
    {
        static void Main(string[] args)
        {
            OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial
catalog=Friends;integrated security=sspi"); // Bağlantı nesnemiz tanımlanıyor.
            string sqltext="Select * From Deneme"; // Tablodaki tüm verileri çekicek sql ifademiz.
            OleDbDataAdapter da=new OleDbDataAdapter(sqltext,con); // OleDbDataAdapter nesnemiz oluşturuluyor.
            DataSet ds=new DataSet(); // DataSet bağlantısız katman nesnemiz oluşturuluyor.
            da.Fill(ds,"Makale"); // DataSet nesnemiz, tablomuza ait veriler ile dolduruluyor.
            /* Tablodaki alanlara ait temel bilgileri edinmek için foreach döngüsünü kullanıyoruz. 0 indisli tablomuz yani deneme
            tablomuza ait tüm alanlar tek tek DataColumn tipindeki c nesnemiz ile dolaşıyoruz. */
            foreach(DataColumn c in ds.Tables[0].Columns)
            {
```

```

        Console.WriteLine("_ "+c.ColumnName.ToString()+"_"); // Alanın adı.
        Console.WriteLine("Alan genişliği _ "+c.MaxLength.ToString()); /* Alan text değer içeriyorsa maksimum uzunluk. */
        Console.WriteLine("Veri türü _ "+c.DataType.ToString()); // Alanın veri türü.
        Console.WriteLine("Null durumu _ "+c.AllowDBNull.ToString()); // Alanın null değer içerebilip içermeyeceği.
        Console.WriteLine("");
    }
}
}
}

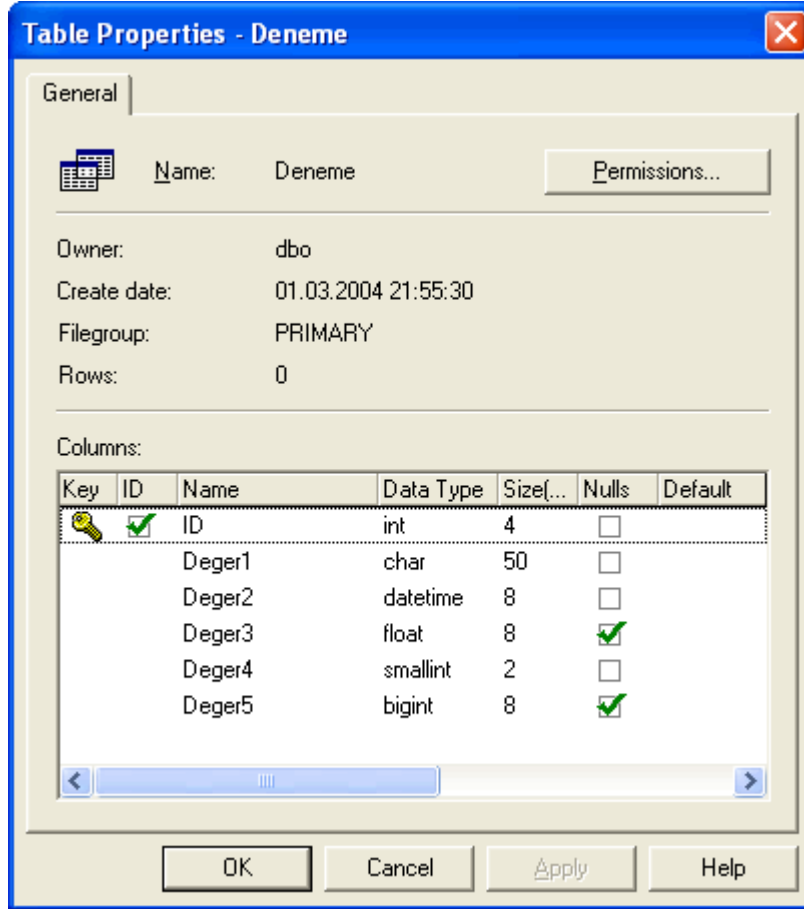
```

Bu kodlarda ne yaptığımızı kısaca anlatalım. Öncelikle, Sql sunucumuzda yer alan Friends isimli veritabanına OleDb veri sağlayıcısı üzerinden bir bağlantı hattı açıyoruz. Daha sonra, bu veritabanındaki Deneme isimli tabloya ait tüm satırları elde edebileceğimiz sql söz dizimi ile bir OleDbDataAdapter nesnesi oluşturuyoruz. Bu nesnenin Fill metodunu kullanarak, DataSet nesnemizi ilgili tabloya ait veriler ile dolduruyoruz. Aynı zamanda sql sunucusundaki deneme isimli tabloya ait alan bilgilerinde elde etmiş oluyoruz. Her bir alanın ismini, bu alan text veri içeriyorsa maksimum karakter uzunluğunu, veri tipini ve null değer içerip içermediğini öğrenmek için, bir döngü kuruyor ve bu döngü içerisinde bu alan bilgilerine, DataColumn sınıfından bir nesne örneğini kullanarak erişiyoruz. Uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ettiğimizi görürüz.



Şekil 1. Uygulamanın çalışmasının sonucu.

Oysa sql sunucumuzda yer alan tablomuzu incelediğimizde bilgilerin daha farklı olduğunu görürüz. İlk göze çarpan, alanların null değer içerebilir içeremeyeceğini gösteren AllowDBNull özelliklerinin doğru bir şekilde elde edilememiş olmalarıdır. Tablomuzun sql sunucusundaki asıl görünümü aşağıdaki gibidir. Kodumuz sonucu tüm alanların null değer içerebileceği belirtilmektedir. Ama durum gerçekte böyle değildir. Diğer yandan string tipteki Deger1 alanının maksimum uzunluk değeri 50 olmasına rağmen uygulamamızın ekran çıktısında -1 olarak görülmektedir.



Şekil 2. Deneme tablomuzun yapısı.

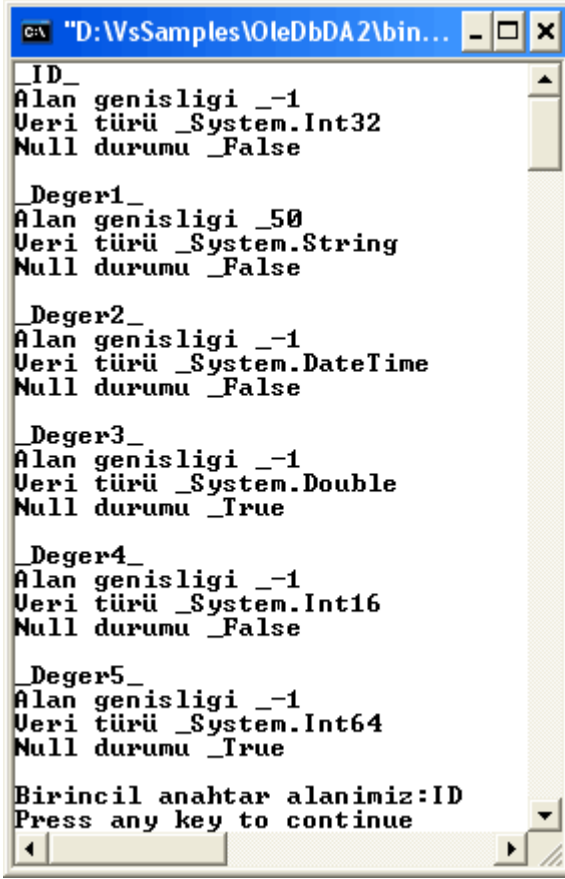
Burada alanlara ait asıl bilgilerin elde edilebilmesi için tabloya ait şema bilgilerinde DataSet nesnemize yüklememiz gerekiyor. İşte bu amaçla, OleDbDataAdapter sınıfına ait FillSchema metodunu kullanırız. FillSchema metodu ilgili tabloya ait alan bilgilerini örneğin Primary Key verisini, bağlantısız katman nesnesine eklememizi sağlar. Bunun için aşağıdaki kodlamayı kullanırız.

```
using System;
using System.Data;
using System.Data.OleDb;

namespace OleDbDA2
{
    class Class1
```

```
{
    static void Main(string[] args)
    {
        OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial
catalog=Friends;integrated security=sspi");
        string sqltext="Select * From Deneme";
        OleDbDataAdapter da=new OleDbDataAdapter(sqltext,con);
        DataSet ds=new DataSet();
        da.FillSchema(ds,SchemaType.Source); // Şema bilgileri ekleniyor.
        da.Fill(ds,"Makale");
        foreach(DataColumn c in ds.Tables[0].Columns)
        {
            Console.WriteLine("_"+c.ColumnName.ToString()+"_");
            Console.WriteLine("Alan genisligi _"+c.MaxLength.ToString());
            Console.WriteLine("Veri türü _"+c.DataType.ToString());
            Console.WriteLine("Null durumu _"+c.AllowDBNull.ToString());
            Console.WriteLine("");
        }
        Console.WriteLine("Birincil anahtar alanımız:"+ds.Tables[0].PrimaryKey[0].ColumnName.ToString());
    }
}
```

Şimdi uygulamamızı çalıştırırsak aşağıdaki sonucu elde ederiz.



Şekil 3. Uygulamanın çalışmasının sonucu.

Artık alanların null değer içerip içermeyeceklerine ait kıstaslar doğru bir şekilde görünmektedir. Aynı zamanda, text bazındaki alanların maksimum uzunluklarında elde edilebilmektedir. (Bu noktada, sayısal alanların genişlik değerlerinin -1 çıkması normaldir. Nitekim DataColumn sınıfının MaxLength özelliği yalnızca text bazlı alanlar için geçerlidir.) Diğer yandan, tablonun birincil anahtar sütununun varlığı elde edilebilmiştir. Primary Key alanının önemini aşağıdaki örnek ile incelemeye çalışalım. Bu örnekte, windows uygulamamızda, bir DataSet nesnesini deneme tablosunun verileri ile dolduruyor ve sonuçları DataGrid kontrolünde gösteriyoruz. İlk etapta FillSchema metodunu kullanmayalım.

```

private void Form1_Load(object sender, System.EventArgs e)
{
    OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated security=sspi");
    string sqltext="Select * From Deneme";
    OleDbDataAdapter da=new OleDbDataAdapter(sqltext,con);
    DataSet ds=new DataSet();
    da.Fill(ds,"Makale");
    dataGrid1.DataSource=ds.Tables["Makale"];
}

```

Uygulamayı çalıştırıp yeni bir satır girelim. ID alanının değerini istediğimiz bir sayı ile değiştirebildiğimizi görürüz.

ID	Deger1	Deger2	Deger3	Deger4	Deger5
2	Deneme1	01.01.2004	4000000	10	151
1454654	Deneme2	01.02.2004	10000	15	48

Şekil 4. ID alanının değerini değiştirebiliyoruz.

Oysaki ID alanımız veri kaynağımızda birincil anahtar olarak tanımlanmıştır ve otomatik olarak artmaktadır. Şimdi uygulamamızda FillSchema metodunu kullanalım.

```
private void Form1_Load(object sender, System.EventArgs e)
{
    OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial catalog=Friends;integrated security=sspi");
    string sqltext="Select * From Deneme";
    OleDbDataAdapter da=new OleDbDataAdapter(sqltext,con);
    DataSet ds=new DataSet();
    da.FillSchema(ds,SchemaType.Source,"Makale");
    da.Fill(ds,"Makale");
    dataGrid1.DataSource=ds.Tables["Makale"];
}
```

Uygulamamızı tekrar çalıştırıp yeni bir satır eklediğimizde ID alanının değerini değiştiremediğimizi ve bu değer yeni bir satırın eklenmesi ile otomatik olarak 1 arttığını görürüz.

ID	Deger1	Deger2	Deger3	Deger4	Deger5
2	Deneme1	01.01.2004	4000000	10	151
3	(null)	(null)	(null)	(null)	(null)

Şekil 5. ID alanına ait kısıtlamanın eklenmesi sonucu.

Bu FillSchema metodunun, veri kaynağındaki tabloya ait ID alanının birincil anahtar kısıtlamasını bağlantısız katmana aktarması sonucu gerçekleşmiştir. Aynı işlemi FillSchema metodunu kullanmadanda gerçekleştirebiliriz. Bunun için, birincil anahtar olacak alana ait temel özelliklerin, ilgili DataTable nesnesinin PrimaryKey özelliğince belirlenmesi gerekir. Yukarıdaki örneğimizi aşağıdaki şekilde geliştirebiliriz.

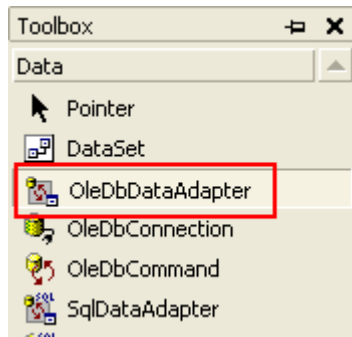
```

private void Form1_Load(object sender, System.EventArgs e)
{
    OleDbConnection con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;initial
catalog=Friends;integrated security=sspi");
    string sqltext="Select * From Deneme";
    OleDbDataAdapter da=new OleDbDataAdapter(sqltext,con);
    DataSet ds=new DataSet();
    da.Fill(ds,"Makale");
    /* Birincil anahtarımız olan ID alanının otomatik artan, null değer içermeyen, 1'den başlayıp 1'er artan ve benzersiz değerler alan
bir alan olduğunu belirtiyoruz. */
    ds.Tables["Makale"].Columns["ID"].AutoIncrement=true; // Alanın değerleri otomatik artacak.
    ds.Tables["Makale"].Columns["ID"].AutoIncrementSeed=1; // Başlama değeri 1 olacak.
    ds.Tables["Makale"].Columns["ID"].AutoIncrementStep=1; // Artış değeri 1 olacak.
    ds.Tables["Makale"].Columns["ID"].AllowDBNull=false; // Alan null değer içermeyecek.
    ds.Tables["Makale"].Columns["ID"].Unique=true; // Alan benzersiz değerler almak zorunda olacak.
    /* Bu tanımlamaların ardından yapmamız gereken, Makale isimli DataTable nesnemiz için PrimaryKey alanının ID alanı olduğunu
belirtmektir. */
    ds.Tables["Makale"].PrimaryKey=new DataColumn[]{ds.Tables["Makale"].Columns["ID"]}; /* ID alanının tanımladığımız
özellikleri ile birlikte, Makale tablosunun birincil anahtarı olacağını belirtiyoruz. */
    dataGrid1.DataSource=ds.Tables["Makale"];
}

```

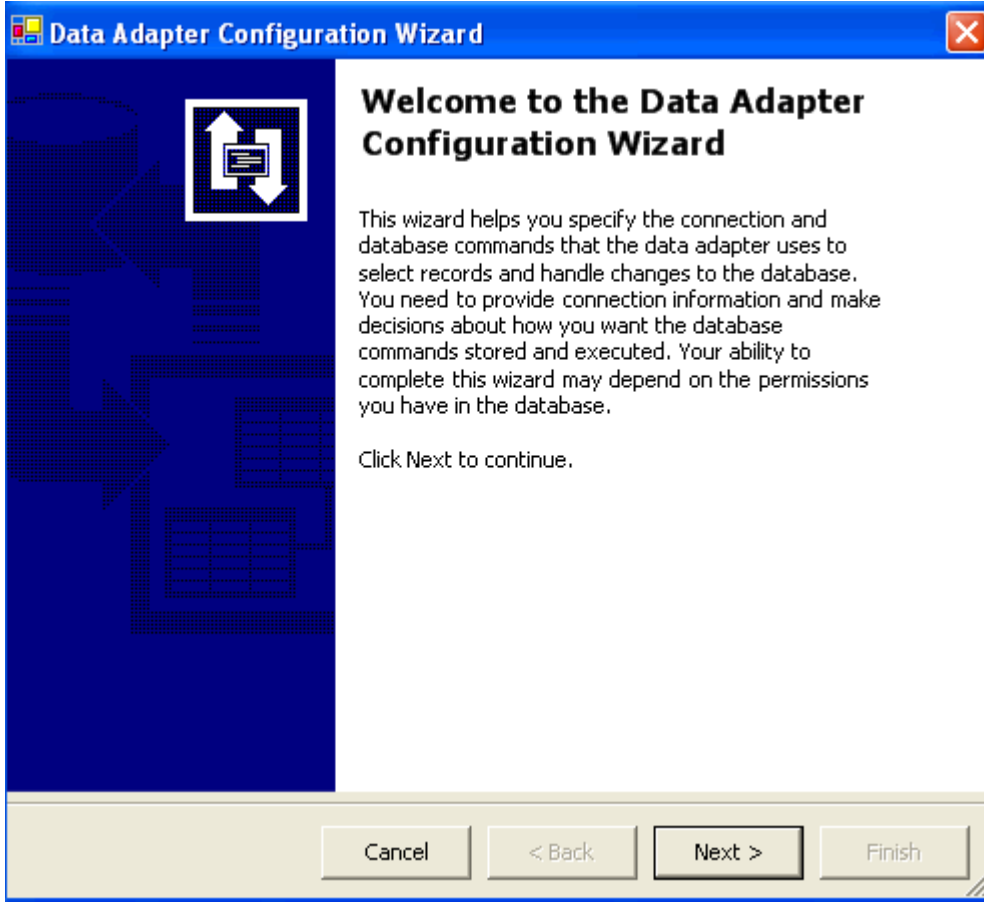
Bu durumda da aynı sonucu elde ederiz. Diğer yandan birbirleri ile ilişkili olan tabloların bu ilişkilerini belirten ForeignKeyConstraint kısıtlamalarının DataSet nesnelere aktarmamız gerekir. Bu konuyu ilerleyen makalelerimizde DataSet kavramını işlerken incelemeye çalışacağız. Şimdi OleDbDataAdapter ile ilgili diğer konularımıza devam edelim.

OleDbDataAdapter sınıfları ile ilgili incelemek istediğim ikinci konu bu nesnelerin, Visual Studio.Net ortamında nasıl oluşturulduğudur. Visual Studio.Net ortamında bu işlem oldukça kolaydır. Bunun için pek çok yöntemimiz var. Bunlardan birisi Server Explorer alanından tabloyu, Form üzerine sürüklemektir. Bu işlem sonucunda, bu tablo için gerekli Connection nesnesi ve DataAdapter nesnesi otomatik olarak oluşturulacaktır. Bir diğer yol ise, OleDbDataAdapter Componentini kullanmaktır. Ben makalemdede, bu ikinci yolu incelemeye çalışacağım. Visual Studio.Net ortamında yeni bir windows uygulaması açın ve Formun üzerine, OleDbDataAdapter componentini sürükleyin.



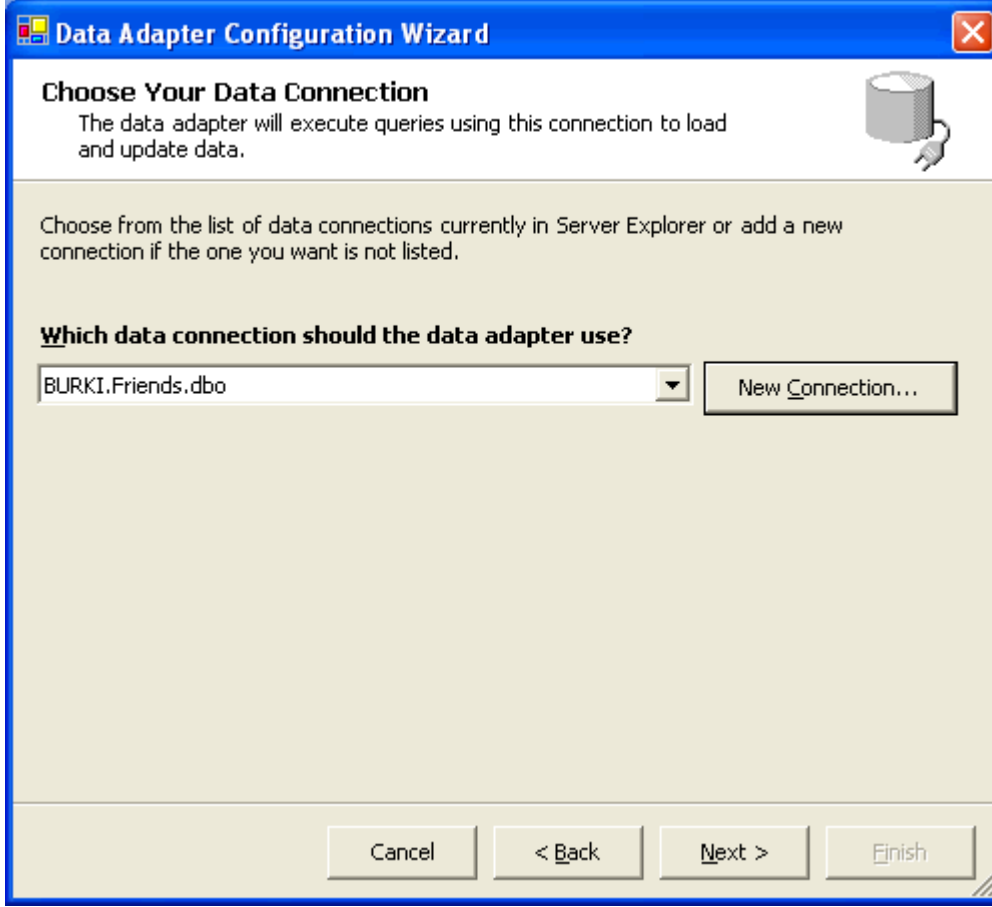
Şekil 6. OleDbDataAdapter aracı diğer ADO.NET araçları gibi Toolbox'ın Data kısmında yer alır.

Bu durumda karşımıza DataAdapter Configuration Wizard penceresi çıkacaktır. Bu pencereyi next butonuna basarak geçelim.



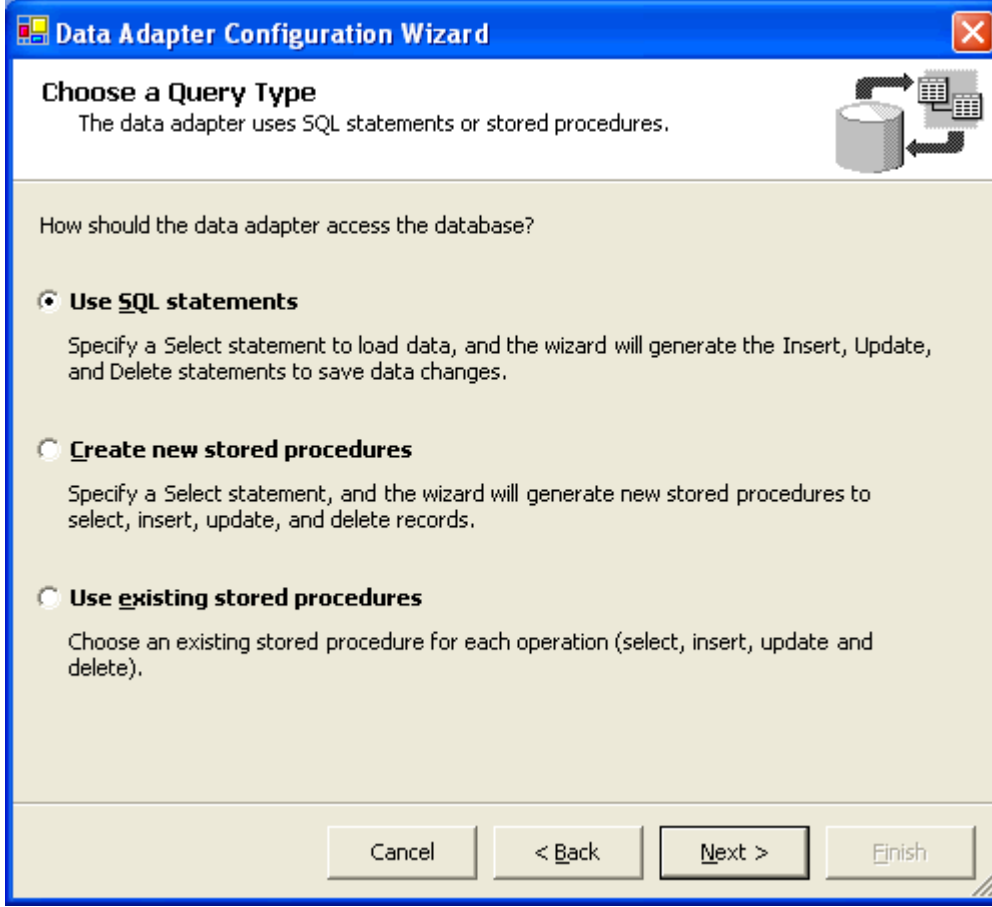
Şekil 7. Başlangıç noktamız.

Bu adımdan sonra, OleDbDataAdapter nesneminiz kullanacağı bağlantı hattı için gerekli bağlantı bilgilerini ayarlarız. Burada halen var olan bir bağlantıyı kullanabileceğimiz gibi New Connection seçeneği ile yeni bir bağlantı bilgisinde oluşturabiliriz. Bu adımda oluşturacağımız bağlantı bilgisi, uygulamamız için gerekli olan OleDbConnection nesnesinin oluşturulmasında kullanılacaktır. Ben burada New Connection diyerek yeni bir bağlantı bilgisi oluşturdum. Artık OleDbDataAdapter nesnemiz bu bağlantı katarını kullanıcak.



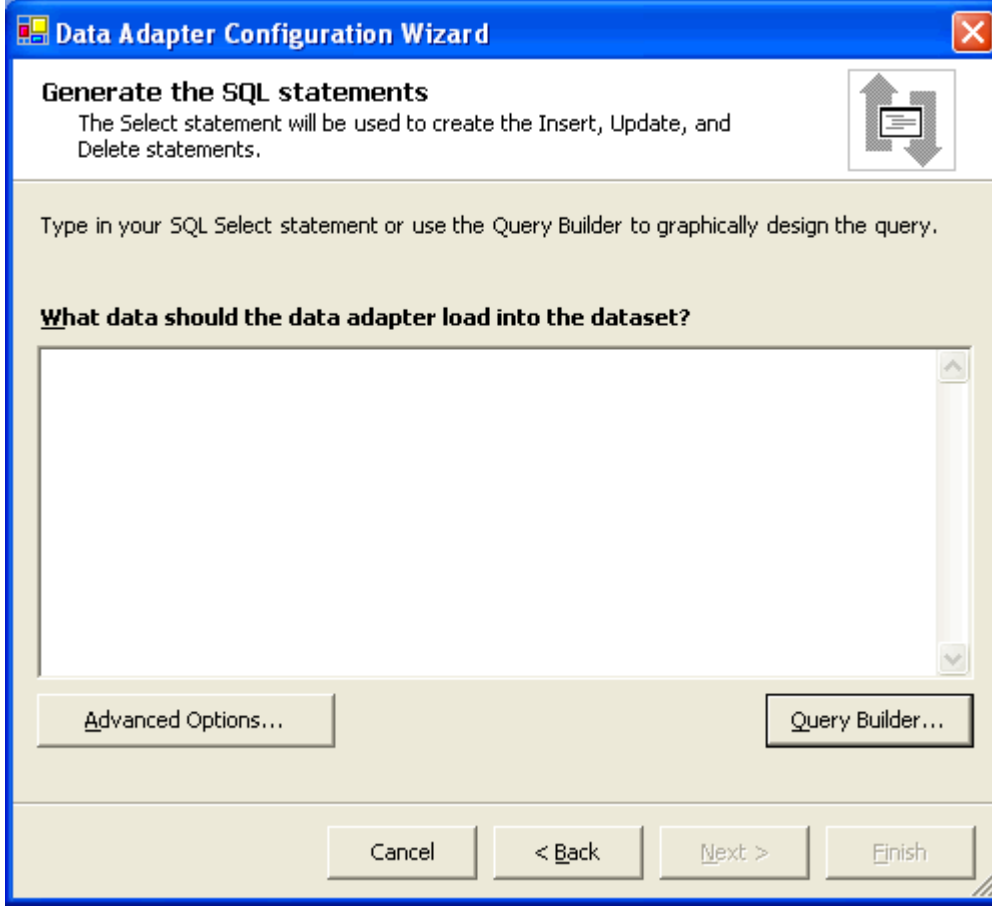
Şekil 8. Bağlantı bilgimizi tanımlıyoruz.

Next ile bu adımda geçtikten sonra, sorgu tipini seçeceğimiz bölüme geliriz. Burada üç seçeneğimiz vardır. Use Sql Statements ile, OleDbDataAdapter nesnesi yardımıyla bilgilerini alacağımız tablo(lar) için gerekli sql ifadelerinin sihirbaz yardımıyla oluşturulmasını sağlarız. Bu işlem sonucunda bizim için gerekli olan SelectCommand, InsertCommand, UpdateCommand ve DeleteCommand özelliklerine ait sql ifadeleri otomatik olarak oluşturulacaktır. Diğer yandan ikinci seçenek ile, bu sql ifadeleri için saklı yordamların oluşturulmasını sağlayabiliriz. Son seçeneğimiz ise, sistemde var olan saklı yordamların kullanılmasını sağlar. Biz şu an için ilk seçeneği seçiyoruz.



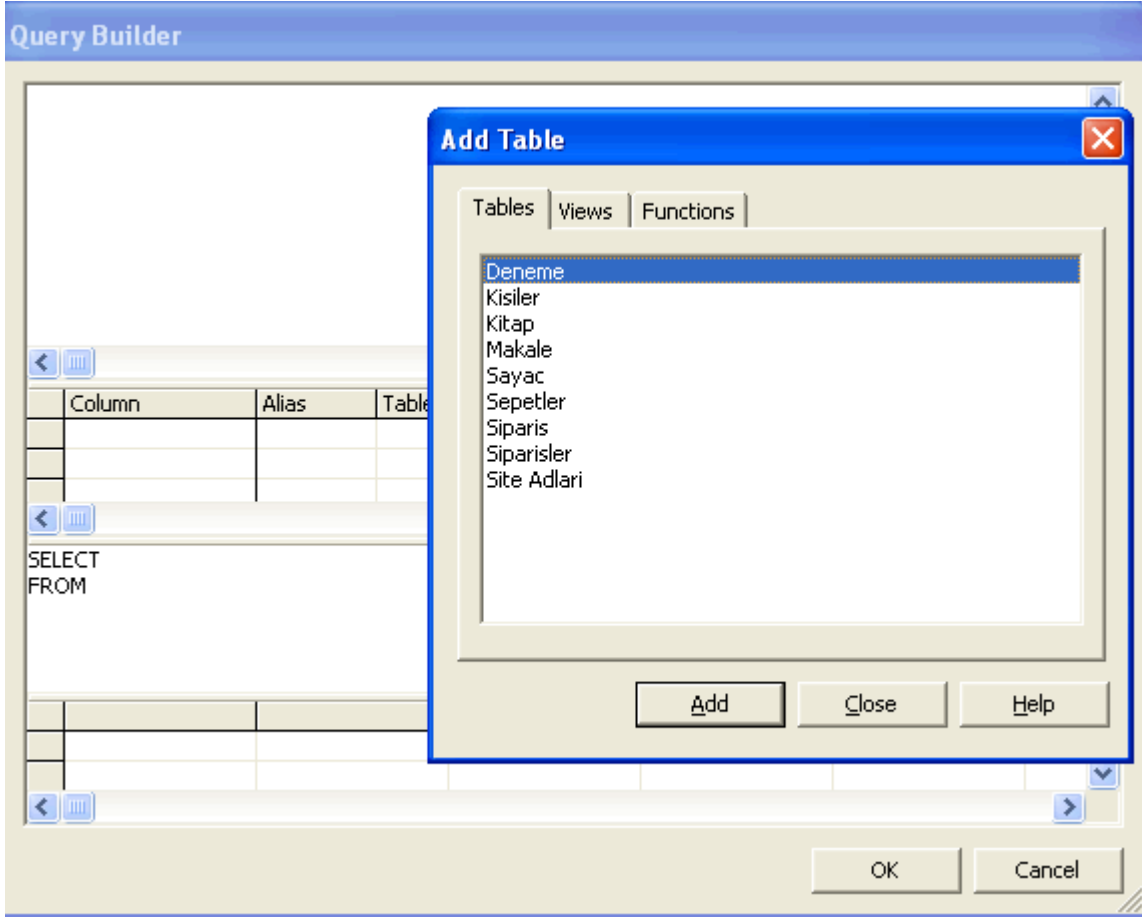
Şekil 9. Sorgu tipinin seçilmesi.

Bu adımdan sonra Select sorgusunu oluşturacağımız bölüm ile karşılaşırız. Burada SelectCommand için kullanılacak sql sorgusunu kendimiz elle yazabileceğimiz gibi Query Builder yardımıyla da bu işlemi daha kolay bir şekilde yapabiliriz. Biz Query Builder seçeneği ile işlemlerimize devam edelim.



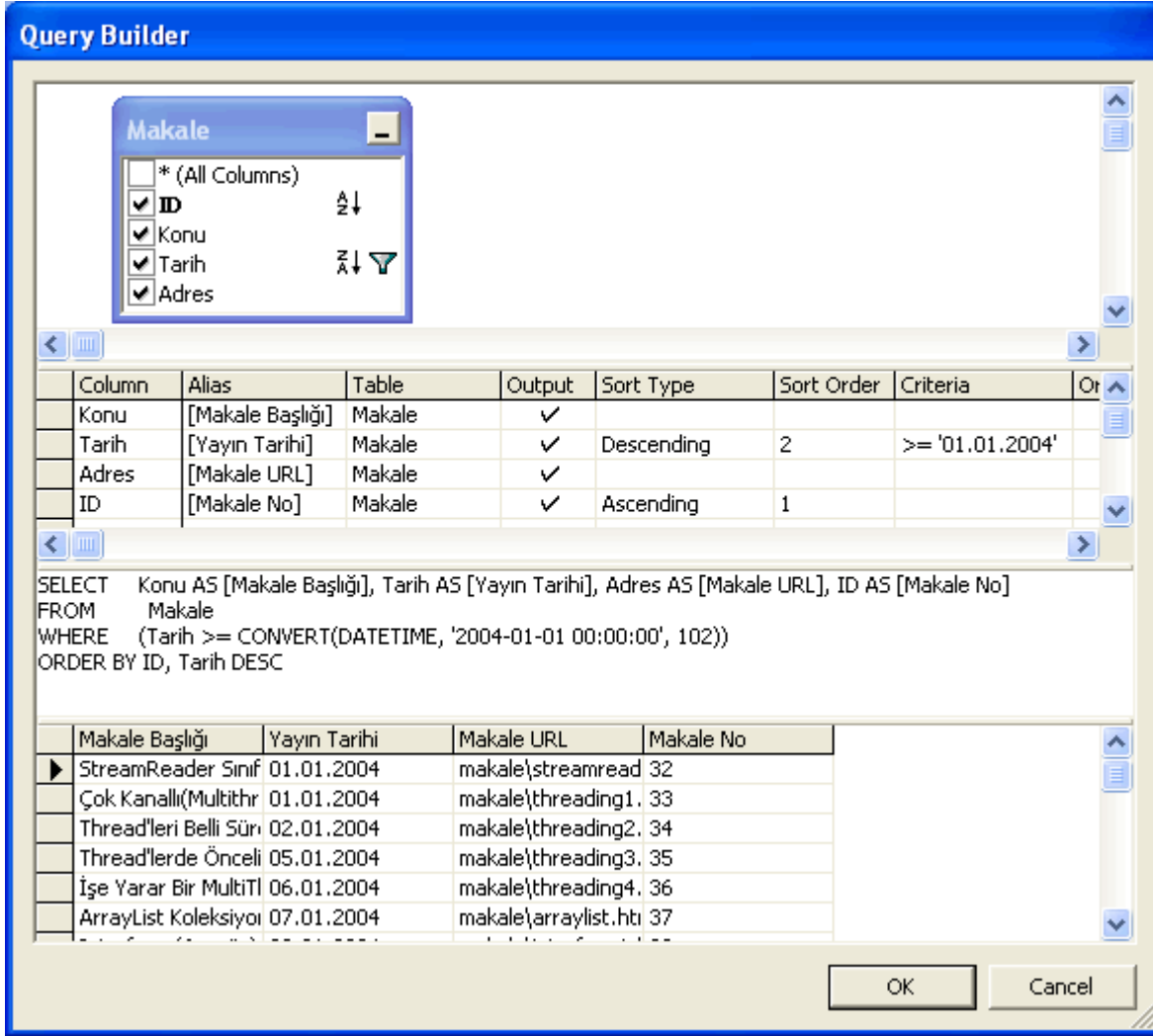
Şekil 10. Sql ifadesinin oluşturulması.

Query Builder kısmında, işlemlerimizin başında belirttiğimiz bağlantı bilgisi kullanılır ve bağlanılan veri kaynağına ilişkin kullanılabilecek tablolar veya görünümeler ekrana otomatik olarak gelir. Tek yapmamız gereken kullanmak istediğimiz tabloyu(tabloları) seçmek ve eklemektir. Daha sonra seçtiğimiz tablo veya tablolardaki hangi alanların select sorgusu ile elde edileceğini ve bağlantısız katmana aktarılacağını belirleriz.



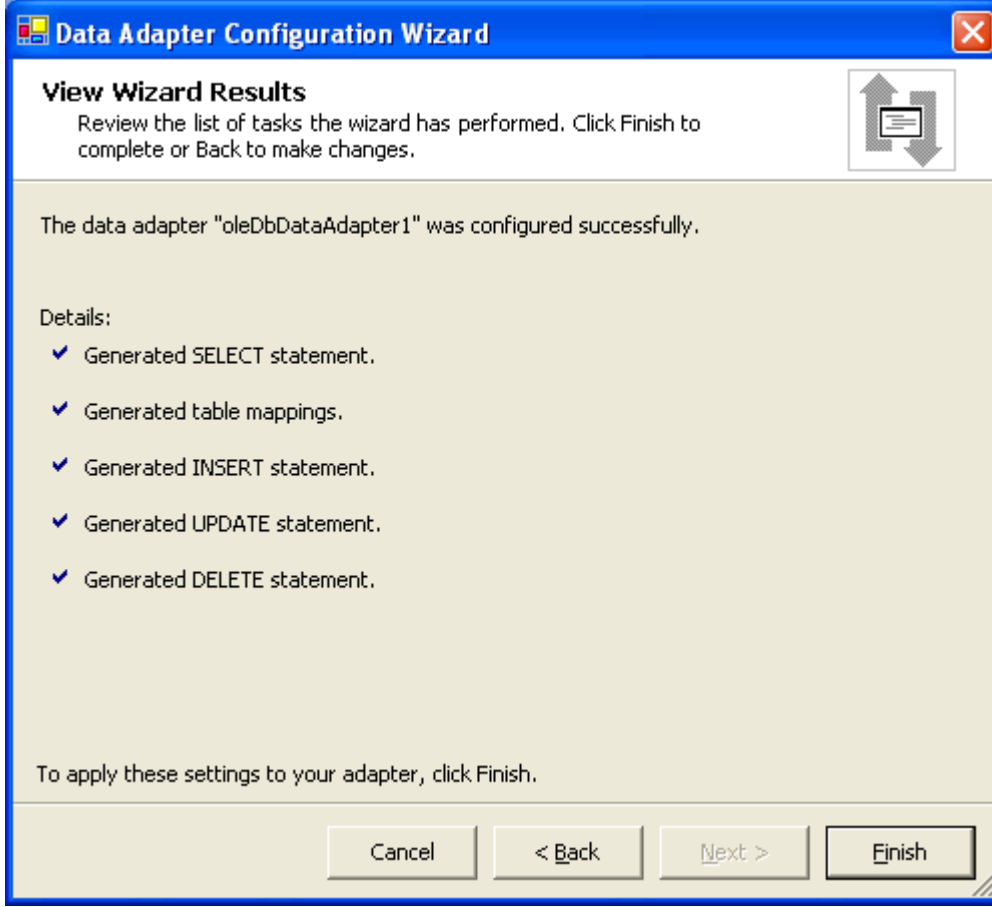
Şekil 11. Tabloların eklenmesi.

Ben burada Makale isimli tablomuzu seçtim ve sorguya ekledim. Daha sonra sadece görünmesini istediğim alanları belirttim. Burada istersek alanların bağlantısız katman nesnesine hangi isimler ile aktarılacağını Alias sütununa yazdığımız değerler ile belirleyebiliriz. Diğer yandan, Sort Type sütununu kullanarak hangi alanlara göre sırlama yapılmasını istediğimizi belirtebiliriz. Ayrıca alanlara ait çeşitli kriterler girebileceğimiz Criteria sütununda yer almaktadır. Bu sütunu kullanmamız Where anahtar kelimesi için bir koşul bildirmek anlamına gelmektedir. Bu işlemler boyunca, sorgu ifademizin otomatik olarak oluşturulduğunu ve geliştirildiğini görürüz. Oluşturulan sorgunun sonucunu görmek için bu alan üzerinde sağ tuşla girdiğimiz menuden Run komutunu verebiliriz. Böylece sorgu sonucu elde edilecek tablo verileri içinde bir öngörünüm elde etmiş oluruz.



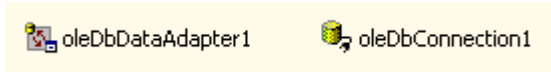
Şekil 12. Query Builder sorgularımızın kolayca oluşturulmasını sağlar.

Bu pencereyi kapttığımızda OleDbDataAdapter nesnemizin, SelectCommand özelliği için gerekli Command nesnesi oluşturulmuş olur. Tekrar Next düğmesine bastığımızda aşağıdaki ekranı elde ederiz. Burada görüldüğü gibi Insert, Update ve Delete sorgularıda otomatik olarak oluşturulmuştur. Ayrıca tablo alanlarımız için kullandığımız eşleştirme işlemleride gerçekleştirilmiş ve TableMappings koleksiyonuda başarılı bir şekilde oluşturulmuştur.



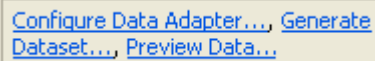
Şekil 13. İşlem Tamam.

Finish'e bastığımızda Formumuzda bir OleDbConnection nesnesinin ve birde OleDbDataAdapter nesnesinin oluşturulmuş olduğunu görürüz.



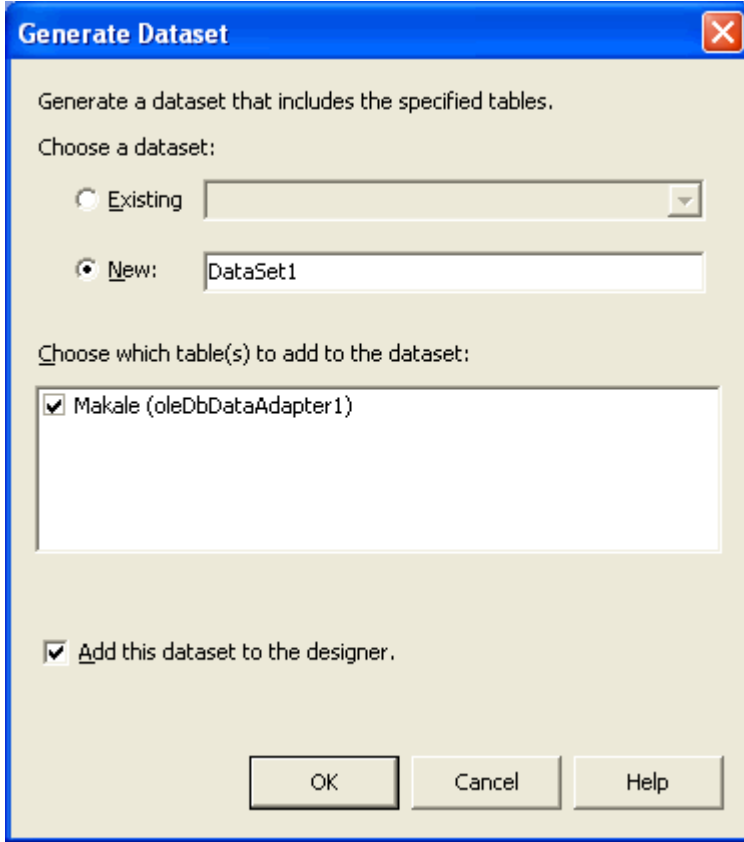
Şekil 14. Nesnelerimiz oluşturuldu.

Şimdi bize, bu OleDbDataAdapter nesnemizin çalışması sonucu elde edilecek verilerin aktarılacağı bir bağlantısız katman nesnesi lazım. Yani bir DataSet nesnesi. Bunun için, OleDbDataAdapter nesnesine ait, Generate DataSet seçeneğini kullanabiliriz.



Şekil 15. Generate DataSet seçeneği, OleDbDataAdapter nesnesinin özelliklerinin altında yer alır.

Bu durumda karşımıza çıkan pencerede var olan bir DataSet'i seçebilir yada otomatik olarak yeni bir tane oluşturulmasını sağlayabiliriz. Bu işlemin ardından DataSet nesnemizde oluşturulduğunu görürüz.

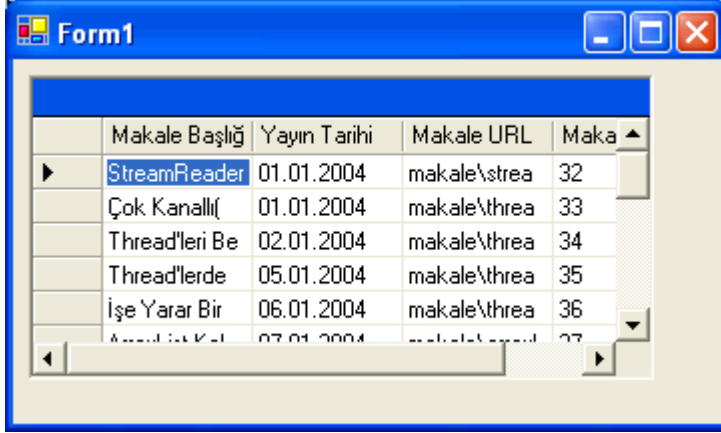


Şekil 16. Generate DataSet penceresi.

Artık yapmamız gerekenler, formumuza bir DataGridView koymak, OleDbDataAdapter nesnemize Fill metodunu uygulamak ve DataSet'imizi doldurmak, son olarakta DataGridView kontrolümüze bu veri kümesine bağlamaktır.

```
private void Form1_Load(object sender, System.EventArgs e)
{
    OleDbDataAdapter1.Fill(dataSet11.Tables["Makale"]);
    dataGridView1.DataSource=dataSet11.Tables["Makale"];
}
```

Uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 17. Uygulamanın çalışmasının sonucu.

Burada yaptığımız işlem ile, OleDbDataAdapter nesnesi ile bağlantısız katmandaki veriler üzerindeki değişiklikleri, veri kaynağına gönderirken Update metodunun kullanacağı UpdateCommand, DeleteCommand, InsertCommand gibi özelliklerin sql ifadelerini otomatik olarak oluşturulmasını sağlamış olduk. Diğer yandan aynı işlevselliği kazanmak için CommandBuilder nesnesinde kullanabiliriz. Bu nesnenin kullanılmasını ve OleDbDataAdapter sınıfına ait Update metodunu bir sonraki makalemizde incelemeye çalışacağız. Hepinize mutlu günler dilerim.

OleDbDataAdapter Sınıfı ve Update Metodu.

Değerli Okurlarım, Merhabalar.

Bu makalemizde, OleDbDataAdapter sınıfının , veriler üzerindeki güncelleme işlemlerinin, veri kaynağına yansıtılması sırasında nasıl bir rol oynadığını ve kullanıldığını incelemeye çalışacağız. Önceki makalelerimizde belirttiğimiz gibi, OleDbDataAdapter nesnesi yardımıyla veri kaynağından, uygulamalarımızdaki bağlantısız katman nesnelere veri kümelerini aktarmak amacıyla Fill metodunu kullanıyorduk. Diğer yandan, bağlantısız katman nesnelerimizin temsil ettiği veriler üzerinde yapılan değişiklikleri veritabanına göndermek istersek, Update metodunu kullanırız.

Update metodu çalışma sistemi açısından oldukça ilgi çekici bir metoddur. Bildiğiniz gibi, DataAdapter nesnelerinin, verilerin güncellenmesi için UpdateCommand, verileri eklemek için InsertCommand, veri silmek için DeleteCommand özellikleri vardır. Uygulamamız çalışırken, bağlantısız katman nesnelerimiz verilerin satırsal bazda durumlarını gösteren bir değer içerir. RowState olarak bilinen bu özellik DataRow sınıfına ait bir özellik olup aşağıdaki tabloda yer alan DataRowState numaralandırıcısı türünden değerlerden birisini almaktadır.

DataRowState Değeri	Açıklama
Added	Yeni bir satır eklendiğini belirtir.
Deleted	Bir satırın silindiğini belirtir.

Modified	Bir satırın düzenlendiğini belirtir.
Detached	Yeni bir satır oluşturulduğunu ama henüz ilgili bağlantısız katman nesnesinin DataRow koleksiyonuna eklenmediğini belirtir.
Unchanged	Satırda herhangi bir değişiklik olmadığını belirtir.

Tablo1. RowState özelliğinin DataRowState numaralandırıcısı tipinden alabileceği değerler.

Buradan yola çıkarsak, Update metodu uygulandığında, OleDbDataAdapter nesnesi, parametre olarak belirtilen DataTable nesnesinin tüm satırlarına bakacaktır. Bu satırlarda yukarıdaki değerleri arayacaktır. Sonuç olarak, Added satırları için, InsertCommand özelliğindeki sql komutunu, Deleted satırlar için DeleteCommand özelliğindeki sql komutunu, Modified satırlar için ise, UpdateCommand özelliğindeki sql komutunu çalıştıracak, böylece uygun güncellemelerin veritabanına en doğru sql ifadeleri ile aktarılmasını sağlayacaktır.

Elbette Update komutunun başarıya ulaşması, Fill metodunun geçerli bir SelectCommand sql komutunu çalıştırmasına bağlıdır. Çünkü, diğer komutların parametreleri bu select sorgusu ile elde edilen tablo alanlarından oluşturulacaktır. Ayrıca, DeleteCommand ve UpdateCommand özelliklerinin sahip olduğu sql komutları Where koşuluna sahiptirler ve bu koşul için çoğunlukla tabloya ait Primary Key(Birincil Anahtar) alanını parametre olarak kullanırlar. Bu sebeple, tablonun birincil anahtara sahip olması önemlidir.

Bahsetmiş olduğumuz güncelleme komutlarını elle programlayabileceğimiz gibi, bu işlemi bizim için basitleştiren CommandBuilder sınıfında kullanabiliriz. Şimdi dilerseniz, OleDbCommandBuilder sınıfı yardımıyla bu işlemin nasıl gerçekleştirileceğini bir örnek üzerinde inceleyelim. Örneğimizde, basit bir sql tablosunu kullanacağız. Öncelikle programımızın kodlarını yazalım.

```
/* global seviyede gerekli nesnelerimizi tanımlıyoruz. Sql sunucusuna bağlantımız için bir OleDbConnection nesnesi, verileri tablodan çekmek ve DataTable nesnemize aktarmak, güncellemelerde aynı dataTable nesnesi üzerinden veri kaynağına göndermek için bir DataAdapter nesnesi, bağlantısız katmanda verilerimizi tutmak için bir dataTable nesnesi ve OleDbDataAdapter nesnemiz için gerekli Update,Delete,Insert komutlarını oluşturacak bir OleDbCommandBuilder nesnesi. */
```

```
OleDbConnection con;
OleDbDataAdapter da;
DataTable dt;
OleDbCommandBuilder cb;
```

```
private void btnDoldur_Click(object sender, System.EventArgs e)
```

```
{
```

```
    con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;database=Friends;integrated security=sspi"); /*
```

```
Bağlantımız oluşturuluyor. */
```

```
    da=new OleDbDataAdapter("Select * From Kisiler",con); /* DataAdapter nesnemiz, select sorgusu ile birlikte oluşturuluyor. */
```

```
    dt=new DataTable("Kisiler"); /*DataTable nesnemiz oluşturuluyor. */
```

```

da.Fill(dt); /* DataTable nesnemizin bellekte gösterdiği alan Kisiler tablosundaki veriler ile dolduruluyor. */
dgKisiler.DataSource=dt; /* DataGrid kontrolümüz, bağlantısız katmandaki verileri işaret eden DataTable nesnemize bağlanıyor.
*/
}

```

```

private void btnGuncelle_Click(object sender, System.EventArgs e)

```

```

{
    try
    {
        cb=new OleDbCommandBuilder(da); /* CommandBuilder nesnemiz , OleDbDataAdapter nesnemiz için oluşturuluyor.
        CommandBuilder'a ait new yapılandırıcısı parametre olarak aldığı OleDbDataAdapter nesnesinin SelectCommand özelliğindeki sql
        komutuna bakarak gerekli diğer UpdateCommand,DeleteCommand ve InsertCommand komutlarını oluşturuyor. */

```

```

        da.Update(dt); /* DataTable'daki değişiklikler Update metodu ile, veritabanına gönderiliyor. */
    }
    catch(Exception hata)
    {
        MessageBox.Show(hata.Message.ToString());
    }
}

```

Uygulamamızı çalıştırdığımızda ve Doldur isimli butona tıkladığımızda, tablomuza ait verilerin dataGrid kontrolüne yüklendiğini görürüz.



Şekil 1. Fill metodunun çalıştırılması sonucu.

Şimdi yeni bir satır ekleyip bir kaç satır üzerinde değişiklik yapalım ve başka bir satırda silelim.



Şekil 2. Update komutunu çalıştırmadan önceki hali.

Güncelle başlıklı butona tıkladığımızda, OleDbCommandBuilder nesnemiz, OleDbDataAdapter nesnemiz için gerekli olan komutları oluşturur. Daha sonra Update metodu çalıştırılmaktadır. Update tablomuzda yapmış olduğumuz düzenleme, silme ve ekleme işlemlerini görmek için, DataTable nesnemizin DataRow koleksiyonundaki her bir DataRow nesnesi için RowState özelliklerinin değerlerine bakar ve uygun olan sql komutlarına bu satırlardaki değerleri parametreler vasıtasıyla aktararak veritabanının güncellenmesini sağlar. Bu noktadan sonra veritabanımızdaki tablomuzla baktığımızda bağlantısız katman nesnesinin işaret ettiği bellek alanındaki tüm değişikliklerin yansıtıldığını görürüz.

Start Page		Form1.cs [Design]	Form1.cs	dbo.Kisiler : T... (BURKLI.Friends)	
	KisiID	Ad	Soyad	DogumTarihi	Meslek
▶	1000	Burak Selim	Senyurt	12.04.1976	Mat.Muh.
	1002	Baska	Birisi	01.01.2004	bil
	1003	Ekelen1	EklenenSoyad	01.01.2001	Diger
*					

Şekil 3. Tabloya yansıtılan değişiklikler.

Dilerseniz CommandBuilder nesnemizin bizim için oluşturmuş olduğu komutların nasıl sql ifadeleri içerdiğini inceleyelim. Bu amaçla, OleDbCommandBuilder sınıfına ait aşağıda prototipleri belirtilen metodları kullanacağız.

Metod	Prototipi
GetInsertCommand	public OleDbCommand GetInsertCommand();
GetDeleteCommand	public OleDbCommand GetDeleteCommand();
GetUpdateCommand	public OleDbCommand GetUpdateCommand();

Tablo 2. OleDbCommandBuilder için Get metodlar.

Dikkat edecek olursanız tüm bu metodlar geriye OleDbCommand sınıfı türünden bir nesne değeri döndürmektedir.

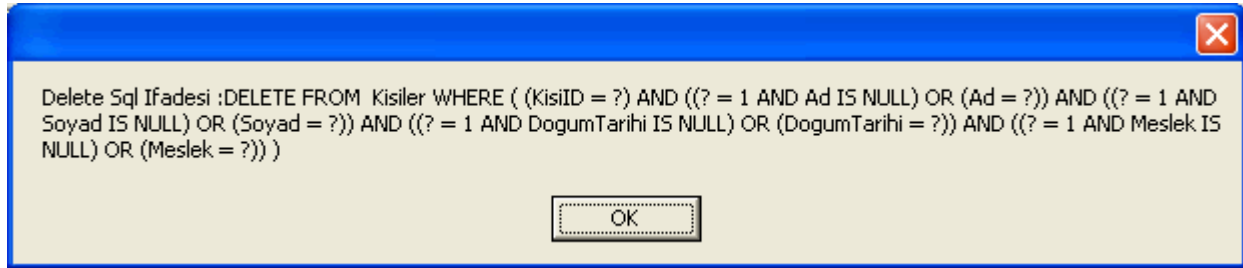
Uygulamamızdaki btnGuncelle kodlarını aşağıdaki gibi düzenlediğimizde, OleDbCommandBuilder nesnesinin, OleDbDataAdapter nesnesi için oluşturmuş olduğu komutları görebiliriz.

```
OleDbCommand cmdInsert=new OleDbCommand();
cmdInsert=cb.GetInsertCommand();
MessageBox.Show("Insert Sql Ifadesi :"+cmdInsert.CommandText.ToString());
```

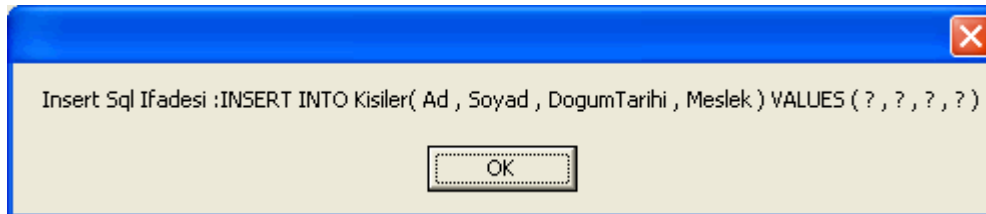
```
OleDbCommand cmdDelete=new OleDbCommand();
cmdDelete=cb.GetDeleteCommand();
MessageBox.Show("Delete Sql Ifadesi :"+cmdDelete.CommandText.ToString());
```

```
OleDbCommand cmdUpdate=new OleDbCommand();
cmdUpdate=cb.GetUpdateCommand();
MessageBox.Show("Update Sql Ifadesi :"+cmdUpdate.CommandText.ToString());
```

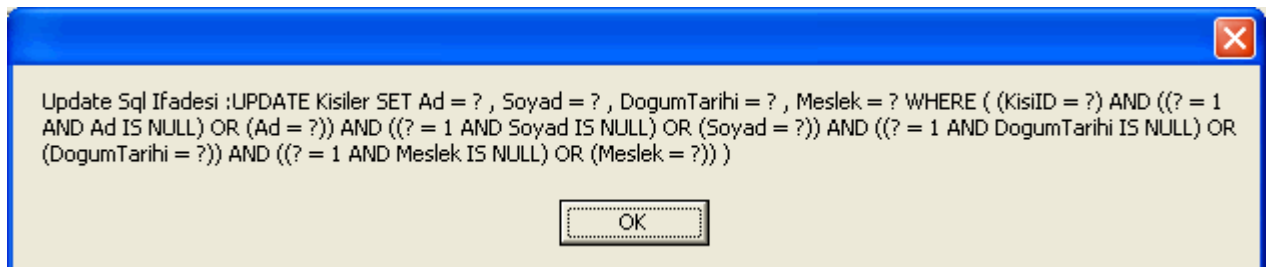
Şimdi uygulamamızı çalıştıralım.



Şekil 4. CommandBuilder nesnemizin oluşturduğu Delete sql komutu.



Şekil 5. CommandBuilder nesnemizin oluşturduğu Insert sql komutu.



Şekil 6. CommandBuilder nesnemizin oluşturduğu Update sql komutu.

Görüldüğü gibi işlem bu kadar basittir. Ancak dilersek CommandBuilder nesnesini kullanmayıp, DataAdapter nesnemiz için gerekli sql komutlarını kendimizde yazabiliriz. Bu biraz daha uzun bir yöntem olmakla birlikte, daha çok kontrole sahip olmamızı sağlar. Ayrıca her programlama dilinde olduğu gibi, işleri böylesine kolaylaştırıcı nesneler performans kaybına neden olabilmektedir. Bu nedenlerden ötürü, OleDbDataAdapter nesnemizin ihtiyaç duyduğu komutları kendimiz yazmak isteyebiliriz. Burada önemli olan nokta gerekli parametrelerin doğru bir şekilde oluşturulmasıdır. Şimdi yukarıda CommandBuilder sınıfı yardımıyla geliştirdiğimiz uygulamayı yineleyelim.

```
OleDbConnection con;
```

```
OleDbDataAdapter da;
```

```
DataTable dt;
```

```
private void btnDoldur_Click(object sender, System.EventArgs e)
```

```
{
```

```
    con=new OleDbConnection("Provider=SQLOLEDB;data source=localhost;database=Friends;integrated security=sspi");
```

```
    da=new OleDbDataAdapter("Select * From Kisiler",con);
```

```
    dt=new DataTable("Kisiler");
```

```
    da.Fill(dt);
```

```
    dgKisiler.DataSource=dt;
```

```
}
```

```
private void btnGuncelle_Click(object sender, System.EventArgs e)
```

```
{
```

```
    try
```

```
    {
```

```
        da.InsertCommand=new OleDbCommand("INSERT INTO Kisiler (Ad,Soyad,DogumTarihi,Meslek) VALUES (?,?,?,?)",con);
```

```
        da.InsertCommand.Parameters.Add("prmAd",OleDbType.VarChar,50,"Ad");
```

```
        da.InsertCommand.Parameters.Add("prmSoyad",OleDbType.VarChar,50,"Soyad");
```

```
        da.InsertCommand.Parameters.Add("prmDogum",OleDbType.Date,8,"DogumTarihi");
```

```
        da.InsertCommand.Parameters.Add("prmMeslek",OleDbType.VarChar,50,"Meslek");
```

```
        da.UpdateCommand=new OleDbCommand("UPDATE Kisiler SET Ad=?,Soyad=?,DogumTarihi=?,Meslek=? WHERE KisiID=?",con);
```

```
        da.UpdateCommand.Parameters.Add("prmKID",OleDbType.Integer,4,"KisiID");
```

```
        da.UpdateCommand.Parameters.Add("prmSoyad",OleDbType.VarChar,50,"Soyad");
```

```
        da.UpdateCommand.Parameters.Add("prmDogum",OleDbType.Date,8,"DogumTarihi");
```

```
        da.UpdateCommand.Parameters.Add("prmMeslek",OleDbType.VarChar,50,"Meslek");
```

```
        da.UpdateCommand.Parameters.Add("prmKisiID",OleDbType.Integer,4,"KisiID");
```

```
        da.DeleteCommand=new OleDbCommand("DELETE FROM Kisiler WHERE KisiID=?",con);
```

```
da.DeleteCommand.Parameters.Add("prmKisiID",OleDbType.Integer,4,"KisiID");
```

```
da.Update(dt);
```

```
}
```

```
catch(Exception hata)
```

```
{
```

```
    MessageBox.Show(hata.Message.ToString());
```

```
}
```

```
}
```

Burada tanımladığımız komutlar için gerekli parametreleri oluştururken Parameters koleksiyonunun Add metodunun aşağıdaki prototipini kullandık.

public OleDbParameter Add(string *parameterName*,OleDbType *oleDbType*,int *size*, string *sourceColumn*);

Buradaki parametreleri kısaca açıklayacak olursak; ilk parametremiz, komutumuz için kullanacağımız parametre adı. İkinci parametremizde ise tablodaki alanımızın veri tipini belirliyoruz. Buradaki veri tipleri OleDbType türündendir. Üçüncü parametremizde ise alanın büyüklüğünü belirtiyoruz. Son parametremiz ise, sql komutu içindeki bu parametrenin hangi alan için kullanılacağını belirtmektedir ve bu anlamı nedeniyle oldukça önemlidir. Dikkat ederseniz OleDb sınıfında OleDbParameter türündeki parametreleri sql komutları içinde ? ile belirttik. Bu nedenle, parametrelerimizi, ilgili sql komutu nesnesinin OleDbParameter koleksiyonuna eklerken ? sırasına göre tanımlamalıyız. Şimdi uygulamamızı çalıştıralım ve veriler üzerinde aşağıdaki görünen değişiklikleri yapalım.



Şekil 7. Değişikliklerimiz yapılıyor.

Şimdi Güncelle başlıklı butonumuza tıklayalım. Değişikliklerin tanımladığımız sql komutları yardımıyla veritabanınada yansıtıldığını görürüz.

Start Page	Form1.cs [Design]	Form1.cs	dbo.Kisiler : T... (BURKL.Friends)	
KisiID	Ad	Soyad	DogumTarihi	Meslek
1000	1000	Senyurt	12.04.1976	Mat.Muh.
1002	1002	Birisi	01.01.2004	Bil. Muh.
1005	Yeni	Birisi	01.01.1975	Bil Muh.
*				

Şekil 8. Değişikliklerimiz veritabanına yansıtıldı.

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde, OleDbDataAdapter sınıfına ait olayları incelemeye çalışacağız. Şimdilik görüşmek dileğiyle, hepinize mutlu günler dilerim.

OleDbDataAdapter Sınıfı Olayları

Değerli Okurlarım, Merhabalar.

Bu makalemizde, OleDbDataAdapter sınıfının olaylarını incelemeye çalışacağız. OleDbDataAdapter sınıfı aşağıdaki tabloda belirtilen üç önemli olayı içermektedir.

Olay	Prototipi	Açıklama
FillError	public event FillErrorEventHandler FillError;	OleDbDataAdapter'ın fill metodu kullanıldığında oluşabilecek bir hata durumunda bu olay çalışır.
RowUpdating	public event OleDbRowUpdatingEventHandler RowUpdating;	Update metodu çalıştırılarak, veritabanındaki tabloya yapılan değişiklikler (satır ekleme, satır silme, satır güncelleme gibi) gerçekleştirildikten önce bu olay çalışır.
RowUpdated	public event OleDbRowUpdatedEventHandler RowUpdated;	Veritabanında yapılacak olan değişiklikler, Update metodu ile gerçekleştirildikten sonra bu olay çalışır.

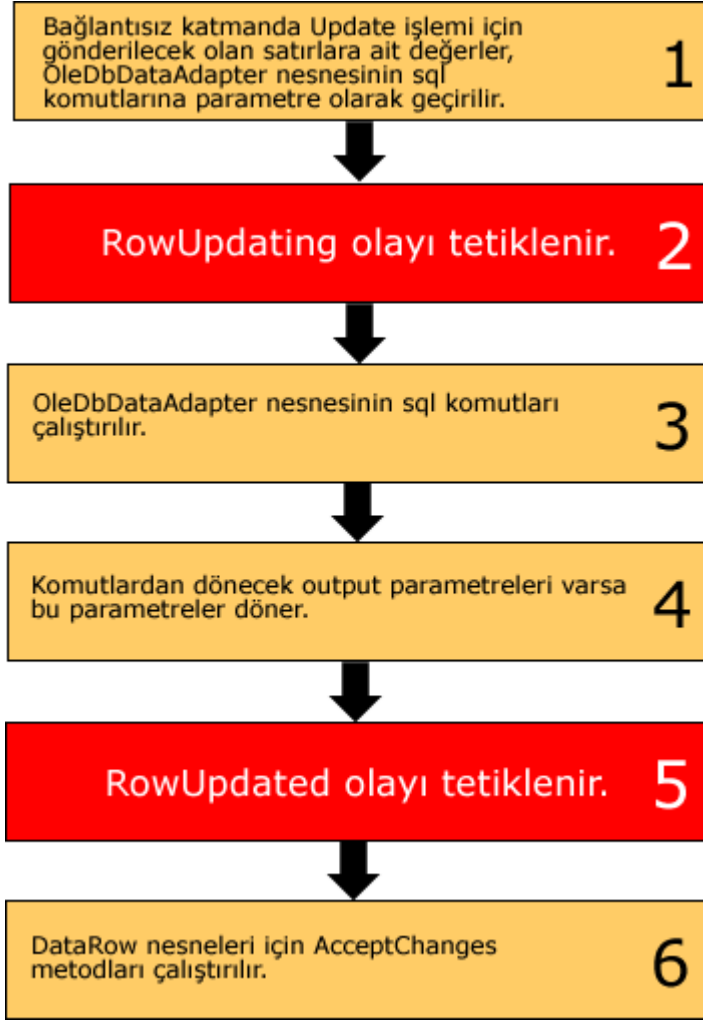
Tablo 1. OleDbDataAdapter olayları.

Şimdi dilerseniz bu olayları kısaca incelemeye çalışalım. RowUpdating olayından başlayalım. Bu olay, OleDbRowUpdatingEventArgs sınıfı türünden bir parametre almaktadır. Bu paramterenin sahip olduğu özellikleri kullanarak, bağlantısız katmandaki veriler, veritabanına yazılmadan önce değişik işlevleri yerine getirme imkanına sahip olmuş oluruz. OleDbRowUpdatingEventArgs sınıfının özellikleri aşağıdaki tabloda yer almaktadır.

OleDbRowUpdatingEventArgs Sınıfı Özellikleri		
Özellik	Görevi	Prototipi
Command	Update metodu çalıştırıldığında çalışacak olan OleDbCommand tipindeki komutu işaret eder. Bu nedenle özelliğin değerinin tipi OleDbCommand'dır.	public new OleDbCommand Command {get; set;}
Row	Update metodu çağırıldığında, veritabanına gönderilecek olan satırı işaret eder. Bu satır DataRow tipindedir ve bu nedenle özelliğin veri tipi DataRow'dur.	public DataRow Row {get;}
Status	Bu özellik ile çalışacak olan komut nesnesinin durumu elde edilir veya değiştirilir. Özellik UpdateStatus numaralandırıcısı türünden bir değeri belirtir. Bu numaralandırıcı Continue, ErrorsOccurred, SkipAllRemainingRows, SkipCurrentRow değerlerinden birisini alır.	public UpdateStatus Status {get; set;}
StatementType	Bu özellik Update metodu ile çalıştırılacak olan sql ifadesini işaret etmektedir. StatementType numaralandırıcısı tipinden bir değeri belirtir. Bu numaralandırıcı sql ifadesinin tipini belirten select, insert, delete ve update değerlerinden birisini alır.	public StatementType StatementType {get;}
TableMapping	Bu özellik, update metodu çalıştırıldığında, bağlantısız katman nesnesindeki tablo haritası ile, veritabanındaki tablo arasındaki eşleştirme ilişkisini DataTableMapping sınıfı örneği olan bir nesne ile ifade eder.	public DataTableMapping TableMapping {get;}
Errors	Update metodu çalıştırıldığında işletilen sql komutunun çalışmasında bir hata oluşması durumunda, oluşan hatayı temsil eden bir özelliktir. Bu sebepler özelliğin tipi Exception'dır.	public Exception Errors {get; set;}

Tablo 2. OleDbRowUpdatingEventArgs Sınıfı Özellikleri

Bu özelliklerin, RowUpdating metodu içinde nasıl işlendiğini örnekler ile incelemeden önce, RowUpdating olayının işleme geçme sürecini ve yerini incelemekte fayda olduğu kanısındayım. RowUpdating olayının, Update metodu içindeki sql komutları çalıştırılıp, gerekli değişiklikler veritabanına yansıtılmadan önce gerçekleştiğini söyleyebiliriz. Aşağıdaki şekil bu konuda bizlere daha iyi bir fikir verecektir.



Şekil 1. RowUpdating ve RowUpdated olaylarının devreye girdiği noktalar.

Şimdi RowUpdating olayını incelemeye başlayalım. Öncelikle basit bir windows uygulaması oluşturalım. Bu uygulamada, Friends isimli veritabanındaki, Kişiler isimli tablomuzun verilerini kullanacağız. Şimdi uygulamamızın başlangıç kodlarını oluşturalım. OleDbDataAdapter nesnemiz için, RowUpdating olayının nasıl eklendiğine dikkatinizi çekmek isterim.

```
/* Sql veritabanındaki Friends isimli veritabanındaki Kisiler isimli tabloya bağlanabilmek için bize gerekli olan nesneleri tanımlıyoruz. Bir OleDbConnection nesnesi, sql veritabanına bağlantı hattı çekmek için; bir OleDbDataAdapter nesnesi, Kisiler tablosundaki verileri, bağlantısız katman nesnemiz olan DataTable'ın veritabanında gösterdiği alana yüklemek ve verilerdeki değişiklikleri veritabanına yazmak için.*/
```

```
OleDbConnection con;
OleDbDataAdapter da;
DataTable dt;
```

```
private void Form1_Load(object sender, System.EventArgs e)
{
```

```

con=new OleDbConnection("provider=SQLOLEDB;data source=localhost;integrated security=sspi;database=Friends"); /*
Bağlantı hattımız oluşturuluyor.*/
da=new OleDbDataAdapter("Select * From Kisiler",con); /* OleDbDataAdapter nesnemiz oluşturuluyor. */
da.RowUpdating+=new OleDbRowUpdatingEventHandler(RowUpdatingOlayi); /*OleDbDataAdapter nesnemiz için, RowUpdating
olayını tanımlıyor ve oluşturuyoruz.*/
dt=new DataTable("Kisiler"); /* DataTable nesnemizin oluşturuluyor. */
}

```

```

private void btnDoldur_Click(object sender, System.EventArgs e)
{
    da.Fill(dt); /* DataTable nesnemizin bellekte işaret ettiği bölge, Kisiler tablosundaki veriler ile dolduruluyor.*/
    dataGrid1.DataSource=dt; /* DataGrid nesnemize veri kaynağı olarak DataTable nesnemiz atanıyor.*/
}
private void btnGuncelle_Click(object sender, System.EventArgs e)
{
    OleDbCommandBuilder cb=new OleDbCommandBuilder(da); /* OleDbDataAdapter nesnemiz için gerekli insert,delete ve update
sql komutlarını otomatik olarak OleDbCommandBuilder yardımıyla oluşturuyoruz.*/
    da.Update(dt); /* DataTable'daki değişiklikler veritabanına gönderiliyor.*/
}

```

```

/* RowUpdating olayımız tetiklendiğinde bu yordamımız çalıştırılacak. */
private void RowUpdatingOlayi(object Sender,OleDbRowUpdatingEventArgs arg)
{
}

```

Şimdi RowUpdating olayımızı incelemeye başlayalım. Örneğin, Row ve Status özelliklerini bir arada inceleyelim. Farzedelimki, KisiID numarası 1000 olan satırının hiç bir şekilde güncellenmesini istemiyoruz. Bu durumu değerlendirebileceğimiz en güzel yer, Update işlemi başarı ile gerçekleşmeden önceki yerdir. Yani RowUpdating metodu.

```

private void RowUpdatingOlayi(object sender,OleDbRowUpdatingEventArgs arg)
{
    if(arg.StatementType==StatementType.Update) /* Eğer şu an yapılan işlem OleDbDataAdapter nesnesinin UpdateCommand
metodunun içerdiği OleDbCommand'ı çalıştıracaksa bu kod bloğu devreye giriyor.*/
    {
        if(arg.Row["KisiID"].ToString()=="1000") /* Şu an işlemde olan satırın KisiID alanının değerine bakıyoruz.*/
        {
            listBox1.Items.Add("1000 nolu kaydı güncelleyemeyiniz.");
            arg.Status=UpdateStatus.SkipCurrentRow; /* SkipCurrentRow ile bu satırın güncellenmesini engelliyoruz.*/
        }
    }
}

```

```

}
}

```

Bu örnek kodlar ile uygulamamızı çalıştırdığımızda, KisiID alanının değerinin 1000 olduğu satırın DataTable üzerinde değiştirilsede, veritabanı üzerinde değiştirilmediğini görürüz. Ancak diğer satırlardaki değişiklikler veritabanına yansıtılır.

KisiID	Ad	Soyad	DogumTarihi	Meslek
1000	Burak S.	Senyurt	12.04.1976	Mat.M
1011	Ahmet A.	Durmasın	01.01.1978	Mat.M
1012	Nihat	Demir	01.01.1975	Mat.M
1013	Ozkan	Çınarcık	04.05.1980	Mat.M
1014	Selim	Demir	05.06.2000	Çocuk

Doldur Guncelle

1000 nolu kaydı güncelleyemessiniz.

Şekil 2. Uygulamanın Çalışması.

Burada Ad alanındaki Burak Selim değerini Burak S. olarak değiştirdik. Bu değişiklik DataTable üzerinde gerçekleşmiştir. Ancak bunu veritabanınada yansıtmak istediğimizde, RowUpdating olayındaki karşılaştırma ifadeleri devreye girecek ve değişiklik veritabanına yansıtılmayacaktır. Visual Studio.NET ortamından tablo içeriğinde baktığımızda bu değişikliğin gerçekleşmediğini görürüz.

	KisiID	Ad	Soyad	DogumTarihi	Meslek
▶	1000	Burak Selim	Senyurt	12.04.1976	Mat.Muh.
	1011	Ahmet A.	Durmasın	01.01.1978	Mat.Muh.
	1012	Nihat	Demir	01.01.1975	Mat.Muh.
	1013	Ozkan	Çınarcık	04.05.1980	Mat.Muh.
	1014	Selim	Demir	05.06.2000	Çocuk
*					

Şekil 3. Değişiklik veritabanındaki tabloya yansıtılmadı.

RowUpdating olayı ile ilgili verilebilecek bir diğer güzel örnek ise, henüz güncellenmiş olan bir satırın başka bir kullanıcı tarafından güncellenmek istenmesi gibi bir durumu kontrol altına almaktır. Bu olayda, o anki satıra ait Orjinal değerlere bakarak, güncellenmek

istenen değerler ile aynı olup olmadığı araştırılır. Böyle bir sonuç çıkarsa kullanıcıya bu satırın zaten güncellendiği tekrar güncellemek isteyip istemeyeceği sorulabilir. Bu işlevi yerine getirmek için RowUpdating olayımızı aşağıdaki gibi şekillendirebiliriz.

```
private void RowUpdatingOlayi(object sender,OleDbRowUpdatingEventArgs arg)
{
    if(arg.StatementType==StatementType.Update)
    {
        string sqlKomutu="SELECT * FROM Kisiler WHERE KisiID='"+arg.Row["KisiID",DataRowVersion.Original]+" AND
Ad='"+arg.Row["Ad",DataRowVersion.Original]+" AND Soyad='"+arg.Row["Soyad",DataRowVersion.Original]+" AND
DogumTarihi='"+arg.Row["DogumTarihi",DataRowVersion.Original]+" AND
Meslek='"+arg.Row["Meslek",DataRowVersion.Original]+"";
        OleDbCommand cmd=new OleDbCommand(sqlKomutu,con);
        con.Open();
        if(cmd.ExecuteNonQuery()==0)
        {
            listBox1.Items.Add("Bu satır zaten güncellenmiş.");
            arg.Status=UpdateStatus.SkipCurrentRow;
        }
    }
}
```

Burada öncelikle bir select sorgusu oluşturuyoruz. Bu sorgu, Update komutu çağırıldığında, OleDbDataAdapter nesnesinin ilgili komutlarına gönderilen satıra ait alanların, en son Fill metodunun çağırılışından sonraki hallerine bakıyor. Eğer aynı güncellemeler başka bir kullanıcı tarafından yapılmış ise, bu güncel satırın o anki değeri ile veritabanındaki aynı olmayacaktır. Dolayısıyla, orjinal değerler değişmiş olduğundan select sorgusunun çalışması sonucu geriye 0 değeri dönecektir. Bu başka bir kullanıcının bu satırı güncellediğini göstermektedir. Bu halde iken kullanıcı uyarılır. İstersek buraya, bir soru kutucuğu açarak kullanıcının bu satırı tekrardan güncellemek isteyip istemediği sorulabilir. Ben bunun geliştirilmesini siz değerli okurlarıma bırakıyorum. If döngümüz içindede bu satır eğer daha önceden güncellenmiş ise, SkipCurrentRow değerini, OleDbRowUpdatingEventArgs sınıfının Status özelliğine atayarak bu satırın güncellenmemesini sağlıyoruz. Şimdi uygulamamızı çalıştırıp deneyelim. Bunun için aynı programı kendi bilgisayarınızda iki kez açmanız yeterli olacaktır.

Form1 (Top)

KisiID	Ad	Soyad	DogumTarihi	Meslek
1000	Burak Selim	Senyurt	12.04.1976	Mat.Muh.
1012	Nihat Ali	D.	01.01.1975	Mat.Muh.
1014	Selim Sahin	Demir	05.01.1978	Mat.Muh.
1016	Ali Dursun	Kel	01.05.1980	Bil.Muh.
1021	Cumhur			
1022	Tahir			

Form1 (Bottom)

KisiID	Ad	Soyad	DogumTarihi	Meslek
1000	Burak Selim	Senyurt	12.04.1976	Mat.Muh.
1012	Nihat Ali	Demir	01.01.1975	Mat.Muh.
1014	Selim Sahin	Demir	05.01.1978	Mat.Muh.
1016	Ali Dursun	Kel	01.05.1980	Bil.Muh.
1021	Cumhur	Tahir	01.05.1975	Öğrenci
1022	Tahir	Kemal	01.04.1958	Dedektif

Form1 (Top) Status:
 Girilen :0
 Silinen :0
 Guncellenen :1

Form1 (Bottom) Buttons:
 Doldur
 Guncelle

Şekil 4. İlk hali.

Önce, soldaki pencerede görülen uygulamada Nihat Ali Demir'in soyadını D. olarak değiştiriyoruz ve Guncelle başlıklı butona basarak bu satırı veri tabanında da güncelliyoruz. Şimdi ekranın sağındaki kullanıcısında aynı soyadını aynı şekilde değiştirmek istediğini düşünelim. Bu amaçla sağdaki programda yine Nihat Ali Demir'in soyadını D. yapıyoruz ve Guncelle başlıklı butona tıklıyoruz. Bu andan itibaren bizim RowUpdating olayına yazdığımız kodlar devreye giriyor. Satırın, Soyad alanının, Fill metodunun çağırılması ile birlikte orjinal değeri halen Demir dir. Şimdi bunu D. nokta yapmak istediğimizde, öncelikle orjinal alan değeri olan Demir select sorgumuza girer. Bu sorgu çalıştığında böyle bir satır bulunamayacaktır. Çünkü Demir, D. ile değiştirilmiştir. Ancak ikinci program fill metodunu bu son güncellemeden sonra çağırmadığı için durumdan habersizdir. Bu nedenle ikinci programın yapmak istediği değişiklik zaten yapılmış olduğundan geri alınacaktır.

KisiID	Ad	Soyad	DogumTarihi	Meslek
1000	Burak Selim	Senyurt	12.04.1976	Mat.Muh.
1012	Nihat Ali	D.	01.01.1975	Mat.Muh.
1014	Selim Sahin	Demir	05.01.1978	Mat.Muh.
1016	Ali Dursun	Kar	01.05.1980	Bil.Muh.
1021	Cumhur	Tahir	01.05.1975	Öğrenci
1022	Tahir	Kemal	01.04.1958	Dedektif
*				

Bu satır zaten güncellenmiş.

Girilen :0
Silinen :0
Guncellenen :0

Doldur Guncelle

Şekil 5. İkinci programın aynı güncellemeyi yapması engellenir.

Bununla birlikte ikinci kullanıcının bu noktadan sonra, D. ismini Demirci olarak değiştirmek istediğini yani farklı bir veri girdiğini farzedelim. Bu değişiklik gerçekleşecektir. Bu durumu şöyle açıklayabiliriz. İkinci program, Demir alanını D. nokta yapmaya çalıştığında, bu güncelleme diğer program tarafından yapılmış olduğundan, satırın güncellenmesi geri alınır. Ancak bu noktada Soyad alanının orjinal değeride değişir ve D. olur. İşte bu nedenle bu noktadan sonra ikinci program bu alanın değerini başka bir değer ile değiştirebilecektir.

Gelelim RowUpdated olayına. Bu olay ise, Şekil 1'de görüldüğü gibi, veritabanına olan güncelleme işlemleri tamamıyla gerçekleştirildikten sonra oluşur ve eklenen, silinen, yada güncellenen her satır için tetiklenir. Bu olayın OleDbDataRowUpdatedEventArgs sınıfı türünden bir parametresi vardır. Bu sınıfın özellikleri OleDbDataRowUpdatingEventArgs sınıfının özellikleri ile aynıdır. Bununla birlikte kullanabileceğimiz ekstradan bir özelliği daha vardır. Bu özellik, RecordsAffected özelliğidir. Bu özellik ile, yapılan güncelleştirmeler sonucu etkilenen satır sayısını elde edebiliriz. Dilerseniz, bu olayı kodumuza uygulayalım. Örneğin yaptığımız güncelleştirmeler sonucu, bu güncelleştirmelerden etkilenen satır sayısını elde etmeye çalışalım. Öncelikle OleDbDataAdapter nesnemize, bu olayımızı ekliyoruz.

```
da.RowUpdated+=new OleDbRowUpdatedEventHandler(RowUpdatedOlayi);
```

Şimdide program kodlarımızı aşağıdaki gibi güncelleyelim.

```
private void btnGuncelle_Click(object sender, System.EventArgs e)
{
```

```

listBox1.Items.Clear();
OleDbCommandBuilder cb=new OleDbCommandBuilder(da);
da.Update(dt);
dt.AcceptChanges();

listBox1.Items.Add("Girilen :"+girilen.ToString());
listBox1.Items.Add("Silinen :"+silinen.ToString());
listBox1.Items.Add("Guncellenen :"+guncellenen.ToString());
}

public int girilen=0,silinen=0,guncellenen=0;

private void RowUpdatedOlayi(object sender,OleDbRowUpdatedEventArgs arg)
{
    if(arg.StatementType==StatementType.Insert)
    {
        girilen+=arg.RecordsAffected;
    }
    else if (arg.StatementType==StatementType.Delete)
    {
        silinen+=arg.RecordsAffected;
    }
    else if (arg.StatementType==StatementType.Update)
    {
        guncellenen+=arg.RecordsAffected;
    }
}

```

Burada yaptığımız son derece basit. RowUpdated olayı, veritabanına girilecek, güncellenecek veya veritabanından silinecek her bir satır için tetiklendiğinden, bu olay yordamı içinde, o anki satır için çalıştırılacak sql ifadesinin ne olduğunu temin ediyoruz. Bunun içinde, OleDbRowUpdatedEventArgs parametresinin StatementType özelliğinin değerine bakıyoruz. Uygun değerlere görede, public integer sayaçlarımızın değerlerini arttırıyoruz. Böylece insert,update ve delete işlemlerinin kaç satıra uygulandığını tespit etmiş oluyoruz.

KisiID	Ad	Soyad	DogumTarihi	Meslek
1000	Burak Selim	Senyurt	12.04.1976	Mat.Muh.
1012	Nihat Ali	Demir	01.01.1975	Mat.Muh.
1014	Selim Sahin	Demir	05.01.1978	Mat.Muh.
1016	Ali Dursun	Kel	01.05.1980	Bil.Muh.
(null)	Cumhur	Tahir	01.05.1975	Öğrenci
(null)	Tahir	Kemal	01.04.1958	Dedektif

Girilen :2
Silinen :1
Guncellenen :3

Şekil 6. Güncelleme sayılarının elde edilmesinin sonucu.

Makalemizde son olarak FillError olayını ele almaya çalışacağım. Bu olay bahsettiğimiz gibi Fill metodu uygulandığında oluşabilecek hatalarda devreye girmektedir.FillError olayı FillEventArgs sınıfı türünden bir parametre alır. FillEventArgs sınıfının, FillError olayı için kullanabileceğimiz özellikleri aşağıdaki tabloda yer almaktadır.

FillEventArgs Özelliği	Açıklama
Continue	Fill metodu ile karşılaşıldığında bir hata olduğu takdirde Continue özelliği kullanılırsa, bu hatalar görmezden gelinerek işleme devam edilir.
Values	Bir hata oluştuğunda bu hata ile ilgili alanların değerlerini belirtir.
DataTable	Hatanın oluştuğu DataTable nesnesine işaret eder.
Errors	Meydana gelen hatayı exception türünden belirtir.

Tablo 3. FillEventArgs Sınıfının Özellikleri

FillError olayının devreye girmesine örnek olarak, veri kaynağındaki veri tiplerinin, .net framework'tekiler ile aynı olmaması durumunu gösterebiliriz.

Böylece geldik bir makalemizin daha sonuna. İlerleyen makalelerimizde Ado.net'in temel kavramlarını incelemeye devam edeceğiz. Hepinize mutlu günler dilerim.

DataTable.Compute Metodu

Değerli Okurlarım, Merhabalar.

Çoğu zaman, uygulamalarımızda T-Sql' in Aggregate fonksiyonlarını kullanarak, belirli sütunlara ait veriler üzerinden, toplam değer, en büyük değer, en küçük değer, ortalama değer vb. gibi sonuçlara ulaşmaya çalışırız. Bu amaçla T-Sql' in Avg, Sum, Count gibi Aggregate fonksiyonlarından yararlanırız. İşte bu makalemizde, bu fonksiyonları, DataTable sınıfının Compute metodu yardımıyla nasıl kullanabileceğimizi incelemeye çalışacağız.

Öncelikle, T-Sql' de yer alan Aggregate fonksiyonlarından kısaca bahsetmekte yarar olduğunu düşünüyorum. Bu fonksiyonların en önemlileri ve kullanışlıları aşağıdaki tabloda yer almaktadır.

Fonksiyon	Prototipi	Açıklama	Dönüş Tipi	Örnek
AVG (Ortalama)	AVG ([ALL DISTINCT] ifade)	Sql sorgusunda belirtilen kritere uyan alanların ortalamasını alır.	int, decimal, money, float	SELECT AVG(Prim) FROM Primler WHERE PerID = 1002124
SUM (Toplam)	SUM ([ALL DISTINCT] ifade)	Sql sorgusunda belirtilen kritere uyan alanların toplam değerini alır.	int, decimal, money, float	SELECT SUM(Prim) FROM Primler
COUNT (Toplam Sayı)	COUNT ({ [ALL DISTINCT] ifade } *)	Satır sayısını verir.	int	SELECT COUNT(*) FROM Primler
MAX (En büyük değer)	MAX ([ALL DISTINCT] ifade)	Belirtilen alana ait sütundaki en büyük değeri verir.	ifade olarak belirtilen tip ile aynıdır.	SELECT MAX(Prim) FROM Primler
MIN (En küçük değer)	MIN ([ALL DISTINCT] ifade)	Belirtilen alana ait sütunlardaki en küçük değeri verir.	ifade olarak belirtilen tip ile aynıdır.	SELECT MIN(Prim) FROM Primler
COUNT_BIG (Toplam Sayı)	COUNT_BIG ({ [ALL DISTINCT] expression } *)	COUNT Fonksiyonu gibi satır sayısını verir. Tek fark dönüş değeridir.	bigint	SELECT COUNT_BIG(*) FROM Primler
STDEV (Standart Sapma)	STDEV (expression)	Belirtilen kritere uyan alanlar için Standart Sapma değerini hesaplar.	float	SELECT STDEV(Alan) FROM Tablo

Tablo 1. Aggregate Fonksiyonları

Bu tip fonksiyonları .net uygulamalarımızda kullanmak için, akla gelen ilk yol Command nesnelerinden yararlanmaktır. Örneğin, Sql sunucumuzda yer alan, Northwind veritabanındaki, Products tablosunu ele alalım. Bu tabloda, Aggregate fonksiyonlarını test edebilmemiz için kullanabileceğimiz alanlar mevcuttur.(UnitPrice, UnitsInStock vb.) Şimdi ilk düşündüğümüz şekilde, yani bir Command nesnesini kullanarak, belli bir gruba ait UnitPrice ve UnitsInStock alanlarının değerleri üzerinde, Aggregate fonksiyonları ile denemeler yapalım. Örneğin basit olması amacıyla bir Console uygulaması geliştirebiliriz.

```

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

namespace Compute
{
    class Class1
    {
        static void Main(string[] args)
        {
            /* Yerel sql sunucumuzdaki Northwind veritabanına bir bağlantı hattı oluşturuyoruz.*/
            SqlConnection con=new SqlConnection("Data Source=localhost;initial catalog=Northwind;Integrated Security=SSPI");
            /* Bu SqlCommand nesnesinin içerdiği sql cümlecği ile, SupplierID değeri 11 olan Products tablosu alanlarının UnitPrice
            değerlerinin toplamını ve kaç satır olduklarının sayısını elde ediyoruz. */
            SqlCommand cmdSum=new SqlCommand("SELECT SUM(UnitPrice),COUNT(SupplierID) FROM Products WHERE
            SupplierID=11",con);
            /* Bu SqlCommand nesnesinin içerdiği sql cümlecği ile, Products tablosundaki UnitPrice alanının değerlerinin ortalamasını
            elde ediyoruz. */
            SqlCommand cmdAvg=new SqlCommand("SELECT AVG(UnitPrice) FROM Products",con);

            /* Bağlantımızı açıyoruz. */
            con.Open();
            SqlDataReader dr; /* SqlDataReader nesnemizi tanımlıyoruz.*/
            dr=cmdSum.ExecuteReader(); /* Komutumuzu çalıştırıp sonuçları bir akım şeklinde SqlDataReader nesnemize
            aktarılacağını belirtiyoruz. */
            /* SqlDataReader akım içinde satır okuyabildiği sürece devam edecek döngümüzü başlatıyoruz ve sorgu sonucu elde edilen
            değerleri ekrana yazdırıyoruz. */
            while(dr.Read())
            {
                Console.WriteLine("Toplam Fiyat {0}, Satır Sayısı {1} ",dr[0],dr[1]);
            }
            dr.Close(); /* SqlDataReader nesnemizi kapatıyoruz. */

            dr=cmdAvg.ExecuteReader(); /* Bu kez SqlDataReader nesnemizi ikinci sorgu cümlecğimizi çalıştıracak SqlCommand
            nesnesi ile oluşturuyoruz. */
            /* SqlDataReader akım içinde satır okuyabildiği sürece devam edecek döngümüzü başlatıyoruz ve sorgu sonucu elde edilen
            değerleri ekrana yazdırıyoruz. */
            while(dr.Read())
            {

```

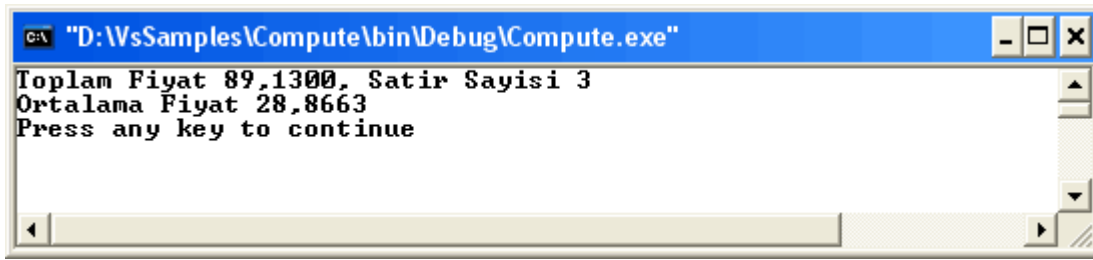
```

        Console.WriteLine("Ortalama Fiyat {0}",dr[0]);
    }

    dr.Close(); /* SqlDataReader nesnemizi kapatıyoruz. */
    con.Close(); /* SqlConnection nesnemizi kapatıyoruz. */
}
}
}

```

Uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 1. SqlCommand ile Aggregate Fonksiyonlarının Kullanımı.

Şimdi gelelim, bu gibi işlemlerin DataTable sınıfı ile nasıl gerçekleştirilebileceğine. Çoğu zaman uygulamalarımızda bağlantısız katman nesneleri ile çalışmaktayız. Bunlardan biriside DataTable nesnesidir. DataTable nesneleri bildiğiniz gibi, veritabanındaki bir tabloya ait içeriğin bellekte tutulduğu bölgeyi işaret ederler. Yada uygulama içerisinde bizim oluşturacağımız bir tablonun bellek görüntüsünü temsil ederler. Her iki haldede, DataTable nesnesinin temsil ettiği bölgede veriler yer alabilir. Bu veriler üzerinde, Aggregate Fonksiyonlarını kullanmak istediğimizde, aşağıda prototipi belirtilen Compute metodunu kullanabiliriz.

public object Compute(string ifade,string filtre);

Compute metodu, belirtilen bir alan için, belirtilen filtreleme mekanizmasının şartları dahilinde, Aggregate Fonksiyonlarının işletilmesinde kullanılır. İlk parametrede SUM, AVG gibi Aggregate fonksiyonlarının kullanıldığı ifade yer alır. İkinci parametre ise karşılaştırma koşulumuzdur. Bu koşul aslında Where koşulunun devamındaki ifadeyi içerir. Dikkat edicek olursanız, Compute metodunun geri dönüş değerinin tipi Object türündendir. Bunun sebebi, çalıştırılan fonksiyonlar sonucu elde edilecek sonuçların veri tipinin tam olarak kestirilememesidir. Aşağıda, Compute metodunun kullanımına ilişkin örnek ifadeler yer almaktadır.

```

object objToplam;
objToplam= Tablo.Compute("Sum(Primler)", "PerID = 8");
object objToplam;
objToplam= Tablo.Compute("Sum(Primler)", "Baslangic > 1/1/2004 AND Bitis < 31/1/2004");

```

Şimdi yukarıdaki örneğimizde, Product isimli veritabanına ait verileri bellekte bir DataTable içinde sakladığımızı düşünelim. Şimdi Aggregate fonksiyonlarımızı bu örnek üzerinde kullanalım. Dilerseniz bu sefer, DataTable üzerindeki Compute metodunun sonuçlarını daha kolay izleyebileceğimiz bir Windows uygulaması geliştirelim. Form tasarımı aşağıdakine benzer şekilde olabilir.



Şekil 2. Form tasarımıımız.

Şimdide uygulamamızın kodlarını yazalım.

```
/* SqlConnection, SqlDataAdapter ve DataTable nesnelerimiz tanımlanıyor.*/
```

```
SqlConnection con;
```

```
SqlDataAdapter da;
```

```
DataTable dtProducts;
```

```
private void Form1_Load(object sender, System.EventArgs e)
```

```
{
```

```
    con=new SqlConnection("Data Source=localhost;initial catalog=Northwind;Integrated Security=SSPI"); /* SqlConnection nesnemiz oluşturuluyor ve Northwind veritabanı için bir bağlantı hattı teşkil ediliyor. */
```

```
    da=new SqlDataAdapter("Select * From Products",con); /* SqlDataAdapter nesnemiz Products tablosundaki tüm veriler üzerinde çalışacak şekilde, geçerli bağlantı nesnesi üzerinden oluşturuluyor. */
```

```
    dtProducts=new DataTable("Urunler"); /* Products tablosundaki verilerin bellekte tutulacağı bölgeyi temsil edecek DataTable nesnemiz oluşturuluyor. */
```

```
}
```

```
private void btnDoldur_Click(object sender, System.EventArgs e)
```

```
{
```

```
    da.Fill(dtProducts); /* DataTable nesnemizin bellekte temsil ettiği bölge, SqlDataAdapter nesnemiz ile dolduruluyor. */
```

```
    dgProducts.DataSource=dtProducts; /* DataGrid nesnemiz veri kaynağına bağlanıyor ve Products tablosundaki verileri göstermesi sağlanıyor. */
```

```
}
```

```
private void btnOrtalama_Click(object sender, System.EventArgs e)
```

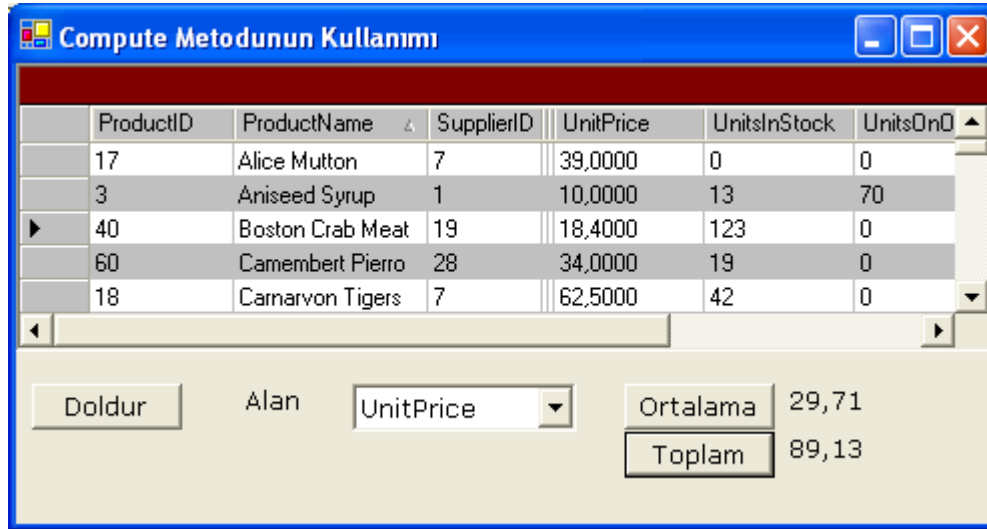
```

{
    double ortalama;
    ortalama=Convert.ToDouble(dtProducts.Compute("AVG("+cmbAlan.SelectedItem.ToString()+")","SupplierID=11")); /* Burada
    kullanıcının seçtiği alana göre SupplierID değeri 11 olanların ortalaması hesaplanıyor. Sonuç noktalı sayı çıkabileceğinden Convert
    sınıfının ToDouble metodu ile Double veri tipine aktarılıyor. */
    lblOrt.Text=ortalama.ToString();
}

private void btnToplam_Click(object sender, System.EventArgs e)
{
    double toplam;
    toplam=Convert.ToDouble(dtProducts.Compute("SUM("+cmbAlan.SelectedItem.ToString()+")","SupplierID=11")); /* Bu kezde
    SupplierID alanının değeri 11 olan alanların Toplam değeri hesaplanıyor. */
    lblToplam.Text=toplam.ToString();
}

```

Uygulamadaki en önemli nokta Compute metodunun kullanım şeklidir. Burada kullanıcının ekrandaki ComboBox kontrolünden seçtiği alana göre işlemler yapılır. Uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz. Bu uygulama daha çok geliştirilebilir. Örneğin koşul ifadesinde kullanıcı tarafından belirlenmesi sağlanabilir. Bu geliştirmeleri siz değerli okurlarımıza bırakıyorum.



Şekil 3. Compute metodu ile Aggregate fonksiyonlarının çalıştırılması.

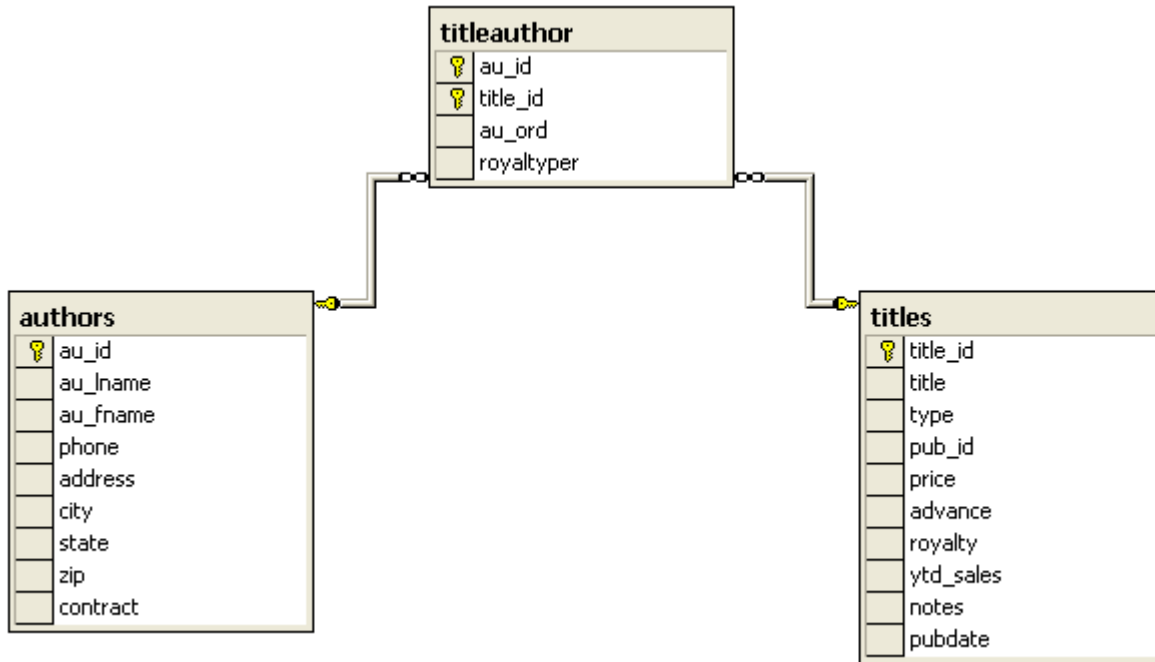
Bu kısa makalemizde DataTable sınıfına ait Compute metodu yardımıyla bağlantısız katman verileri üzerinde Aggregate Fonksiyonlarını kolayca nasıl kullanabileceğimizi incelemeye çalıştık. Umarım siz değerli okurlarım için yararlı bir makale olmuştur. Bir sonraki makalemizde görüşmek dileğiyle, hepinize mutlu günler dilerim.

DataRelation Sınıfı ve Çoğa-Çok (Many-to-many) İlişkiler

Değerli Okurlarım, Merhabalar.

Bugünkü makalemizde, DataRelation sınıfı yardımıyla, veritabanlarındaki many-to-many(Çoğa-çok) ilişkilerin, bağlantısız katmanda nasıl kullanılabildiğini incelemeye çalışacağız. İlişkisel veri tabanı modelinde, tablolar arası ilişkilerde çoğunlukla bire-çok(one-to-many) ilişkilere rastlarız. Ancak azda olsa, çoğa-çok ilişkilerin kullanıldığı durumlarda söz konusudur. Bu ilişkiye örnek olarak çoğunlukla, Sql sunucusunda yer alan Pubs veritabanındaki Authors ve Titles tabloları gösterilir. Bu iki tablo arasındaki ilişki şöyledir; bir yazara ait birden fazla kitap titles tablosunda yer alabilir. Aynı şekilde, bir kitap birden fazla yazar tarafından kaleme alınmış olabilir. Bu bahsedilen iki ilişkide ayrı ayrı bire-çok ilişkilerdir. Yani bir yazarın birden fazla kitabı yazmış olması bire çok ilişki olarak düşünülebilirken, bir kitabın birden fazla yazara ait olmasında bire-çok ilişki olarak gözlemlenebilir.

Ancak, bu iki tablo arasında ilişkiyi bu şekilde yansıtmamız mümkün değildir. Nitekim, bire-çok ilişkilerde, çok ilişkiyi temsil eden tablodaki yabancı anahtar(foreign key), ebeveyn(parent) tabloda unique özellikte bir alana ihtiyaç duyar. Dolayısıyla iki yönlü ilişkinin olduğu authors ve titles gibi tablolar için bu tarz bir ilişkiyi oluşturmak biraz daha farklıdır. Bunun için üçüncü bir tablo kullanılır ve bu tabloda, her iki tablonun primary key alanlarına yer verilir. Aşağıdaki şekil, pubs veritabanında yer alan authors ve titles tabloları için çoğa-çok ilişkiyi sağlayacak bu tarz bir tablonun yapısını ve aralarındaki ilişkiyi göstermektedir.



Şekil 1. titleauthor tablosu yardımıyla çoğa-çok ilişkinin tanımlanması.

Authors tablosunda yer alan her au_id alanı, titleauthors tablosunda yer alır. Aynı durum titles tablosundaki title_id alanı içinde geçerlidir. Böylece, author ve titles tabloları arasındaki çoğa-çok ilişki, titleauthor tablosu üzerinden gerçekleştirilebilmektedir.

Şimdi derseniz, kendimiz çoğa-çok ilişkiye sahip iki tablo ve bu tabloların arasındaki ilişkiyi gerçekleştirecek üçüncü bir ara tabloyu oluşturalım. Örnek olarak, büyük bir yazılım şirketindeki proje elemanlarını ve gerçekleştirilen projeleri ele alabiliriz. Bir proje mühendisi pek çok proje gerçekleştirebileceği gibi, yapılan, halen çalışılan veya planlanan projelerde birden fazla proje mühendiside

görev alabilir. İşte bu çoğa-çok ilişki için gösterebileceğimiz güzel bir örnektir. Bu amaçla sql sunucumuzda aşağıdaki yapılara sahip tabloları oluşturalım.

Table Properties - Muhendisler

General

Name: Muhendisler Permissions...

Owner: dbo

Create date: 31.03.2004 17:06:53

Filegroup: PRIMARY

Rows: 0

Columns:

Key	ID	Name	Data Type	Size...	Nulls	Default
	☒	PersonelID	int	4	☐	
		Bolum	char	10	☐	
		Ad	char	25	☐	
		Soyad	char	25	☐	
		Brans	char	25	☐	

OK Cancel Apply Help

Şekil 2. Projedeki mühendisler için genel bilgileri taşıyan Muhendisler tablosu.

Table Properties - Projeler

General

Name: Projeler [Permissions...](#)

Owner: dbo

Create date: 31.03.2004 17:09:07

Filegroup: PRIMARY

Rows: 0

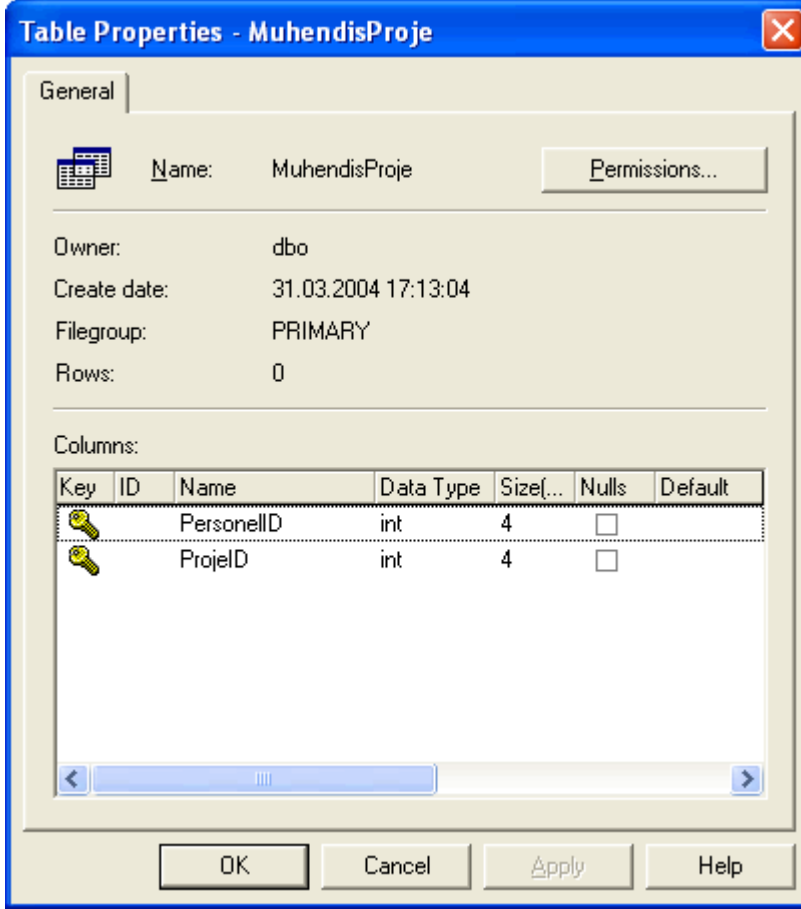
Columns:

Key	ID	Name	Data Type	Size(...)	Nulls	Default
	☒	ProjID	int	4	☐	
		ProjeAdı	char	50	☐	
		ProjeTanımı	char	250	☐	
		Baslangic	datetime	8	☒	
		Bitis	datetime	8	☒	

OK Cancel Apply Help

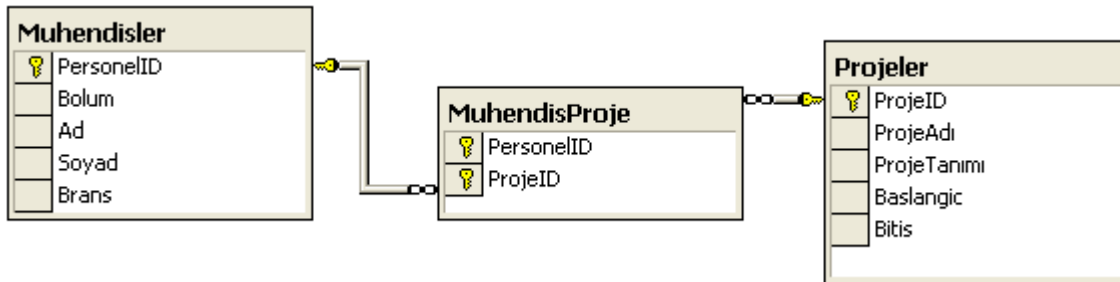
Şekil 3. Projelere ait yüzeysel bilgileri tutacak olan Projeler tablosu.

Son olarakta çokla-çok ilişkiyi taşıyacak ara tablomuz.



Şekil 4. MuhendisProje Tablomuz Çoka-çok ilişkiyi taşıyacak.

MuhendisProje tablosunu oluşturduğumuzda, Muhendisler tablosundan bu tabloya bire-çok ilişki ve yine Projeler tablosundan bu tabloya bire-çok ilişkileri aşağıdaki gibi oluşturmamızda gerekiyor.



Şekil 5. MuhendisProje tablosu üzerinden gerçekleştirilen çoka-çok ilişki.

Gelelim işin .net kısmına. Tasarladığımız bu yapıyı uygulamalarımızda kullanabilmek için, özellikle bağlantısız katman nesneleri üzerinde kullanabilmek için DataRelation sınıfını kullanmamız gerekiyor. Yukarıdaki işlemler ile sql sunucumuzda oluşturduğumuz düzenin aynısını , sistemimizdeki bağlantısız katman uygulamasında gerçekleştirmek istediğimiz senaryoyu göz önüne alalım.

Öncelikle, sahip olduğumuz üç tabloyuda bir DataSet nesnesine aktarmamız gerekiyor. Daha sonra, sql sunucusundaki bu tablolar arasındaki ilişkileri, DataSet içerisindeki tablolarımız arasındada gerçekleştirmemiz gerekli. İşte bu noktada DataRelation sınıfı

devreye giriyor. Önce, Muhendisler tablosundan, MuhendisProje tablosuna olan bire-çok ilişkiyi oluşturuyoruz. Ardından ise, Projeler tablosundan, MuhendisProje tablosuna olan bire-çok ilişkiyi tasarlıyoruz. Bu ilişkilerin DataRelation nesneleri olarak tanımlanmasının ardından, DataSet sınıfının DataRelation nesnelerini taşıyan Relations koleksiyonunada eklenmeleri gerekiyor. İşte bu son adım ile birlikte, veritabanı sunucusundaki çoğa-çok ilişkinin aynısını bağlantısız katman nesnemiz olan DataSet üzerinde de gerçekleştirmiş oluyoruz.

Dilerseniz, yukarıda özetlediğimiz işin uygulamada nasıl gerçekleştirilebileceğini incelemeye çalışalım. Bunu için bir windows uygulaması geliştirebiliriz. Bu uygulamada bir proje mühendisi seçildiğinde, bu mühendisin yer aldığı projeleri ve bu projelerdeki ekip arkadaşlarını gösterecek olan bir uygulama geliştirelim. Bu amaçla aşağıdakine benzer tarzda bir form hazırlayalım.

Şekil 6. Form Tasarımımız.

Sıra geldi uygulamamızın kodlarını yazmaya. Uygulamayı iki kısımda yazarsak daha kolay anlaşılır olacaktır. Öncelikle, bir mühendisi seçtiğimizde bu mühendisin görev aldığı projeleri elde edebileceğimiz kodu uygulamamıza ekleyelim. Bu aşamada uygulamamızın kodları aşağıdaki gibi olacaktır.

```
SqlConnection con;
SqlDataAdapter da;
DataSet ds;
DataTable dtMuhendisler;
DataTable dtProjeler;
DataTable dtMuhendisProje;

private void Form1_Load(object sender, System.EventArgs e)
{
    /* Sql sunucumuza bir bağlantı açıyoruz. */
    con=new SqlConnection("Data source=localhost;initial catalog=Friends;integrated security=SSPI");
```

```

/* Mühendisler, MuhendisProje ve Projeler tablolarını referans edecek DataTable nesneleri ile bu DataTable nesnelerini
bünyesinde barındıracak DataSet nesnemizi oluşturuyoruz.*/
DataSet ds=new DataSet();
dtMuhendisler=new DataTable();
dtProjeler=new DataTable();
dtMuhendisProje=new DataTable();

/* SqlDataAdapter nesnemiz ile ilk aşamada, Muhendisler tablosundaki bilgileri alıyor ve dtMuhendisler DataTable nesnesine
yükliyoruz. */
da=new SqlDataAdapter("Select * From Muhendisler",con);
da.Fill(dtMuhendisler);
/* Tanımladığımız primaryKey alanını, listBox kontrolündeki bir Muhendisi seçtiğimizde bu Muhendise ait satırı bulucak Find
metodunda kullanabilmek için, PersonelID üzerinden oluşturuyoruz. */
dtMuhendisler.PrimaryKey=new DataColumn[] {dtMuhendisler.Columns["PersonelID"]};
/* dtMuhendisler DataTable nesnesini, DataSet in Tables koleksiyonuna ekliyoruz. */
ds.Tables.Add(dtMuhendisler);

/* Şimdi Projeler tablosundaki verileri, dtProjeler DataTable nesnesine ekliyor ve bu dataTable'ıda DataSet nesnemizin Tables
koleksiyonuna ekliyoruz.*/
da=new SqlDataAdapter("Select * From Projeler",con);
da.Fill(dtProjeler);
ds.Tables.Add(dtProjeler);

/* Sıra MuhendisProje tablosundaki verilerin eklenmesinde. */
da=new SqlDataAdapter("Select * From MuhendisProje",con);
da.Fill(dtMuhendisProje);
ds.Tables.Add(dtMuhendisProje);

/* Şimdi işin önemli kısmı. Muhendisler tablosundan MuhendisProje tablosuna olan bire-çok ilişkiyi DataSet nesnemizin Relations
koleksiyonuna bir DataRelation nesnesi olarak ekliyoruz.*/

ds.Relations.Add("Muhendis_MuhendisProje",dtMuhendisler.Columns["PersonelID"],dtMuhendisProje.Columns["PersonelID"],false);

/* Burada ise, Projeler tablosunda, MuhendisProje tablosuna olan bire-çok ilişkiyi tanımlıyor ve DataSet nesnemizin Relations
koleksiyonuna DataRelation olarak ekliyoruz.*/
ds.Relations.Add("Projeler_MuhendisProje",dtProjeler.Columns["ProjeID"],dtMuhendisProje.Columns["ProjeID"],false);

/* lbMuhendisler ListBox kontrolünün dtMusteriler DataTable'ındaki verileri göstereceğini belirtiyoruz.*/

```

```
lbMuhendisler.DataSource=dtMuhendisler;
```

```
/* Buradaki satırlarda, Form üzerindeki lbMuhendisler ListBox kontrolü için, Muhendislerin adlarını gösterecek DisplayMember ve her bir mühendisin PersonelID alanının değerini indis olarak alıcak ValueMember özelliklerini belirliyoruz.*/
```

```
lbMuhendisler.DisplayMember=dtMuhendisler.Columns["Ad"].ToString();
```

```
lbMuhendisler.ValueMember=dtMuhendisler.Columns["PersonelID"].ToString();
```

```
}
```

```
private void btnGetir_Click(object sender, System.EventArgs e)
```

```
{
```

```
lbProjeler.Items.Clear();
```

```
/* Öncelikle lbMuhendisler ListBox kontrolünden seçilen Mühendise ait PersonelID alanının değerini alıyor ve DataTable sınıfının Rows koleksiyonunun Find metodu yardımıyla bu PrimaryKey değerini içeren satırı dtMuhendisler tablosundan buluyor ve DataRow nesnesine aktarıyoruz.*/
```

```
DataRow dr;
```

```
dr=dtMuhendisler.Rows.Find(lbMuhendisler.SelectedValue);
```

```
/* Foreach döngüsünde ilk aşamada, seçilen Muhendisin çocuk satırlarını(child rows) MuhendisPersonel tablosundan alıyoruz. Bir veya birden fazla satır geldiğini düşünürsek her bir satır içinde, Projeler ve MuhendisProje tabloları arasındaki ilişkiyi kullanarak, GetParentRow metodu yardımıyla, Mühendislerin çalıştığı Projelere ait satırları elde ediyoruz. Sonrada bu satırlardaki ProjeAdi alanının değerini lbPorjeler isimli ListBox kontrolümüze ekliyoruz.*/
```

```
foreach(DataRow drProjeNo in dr.GetChildRows("Muhendis_MuhendisProje"))
```

```
{
```

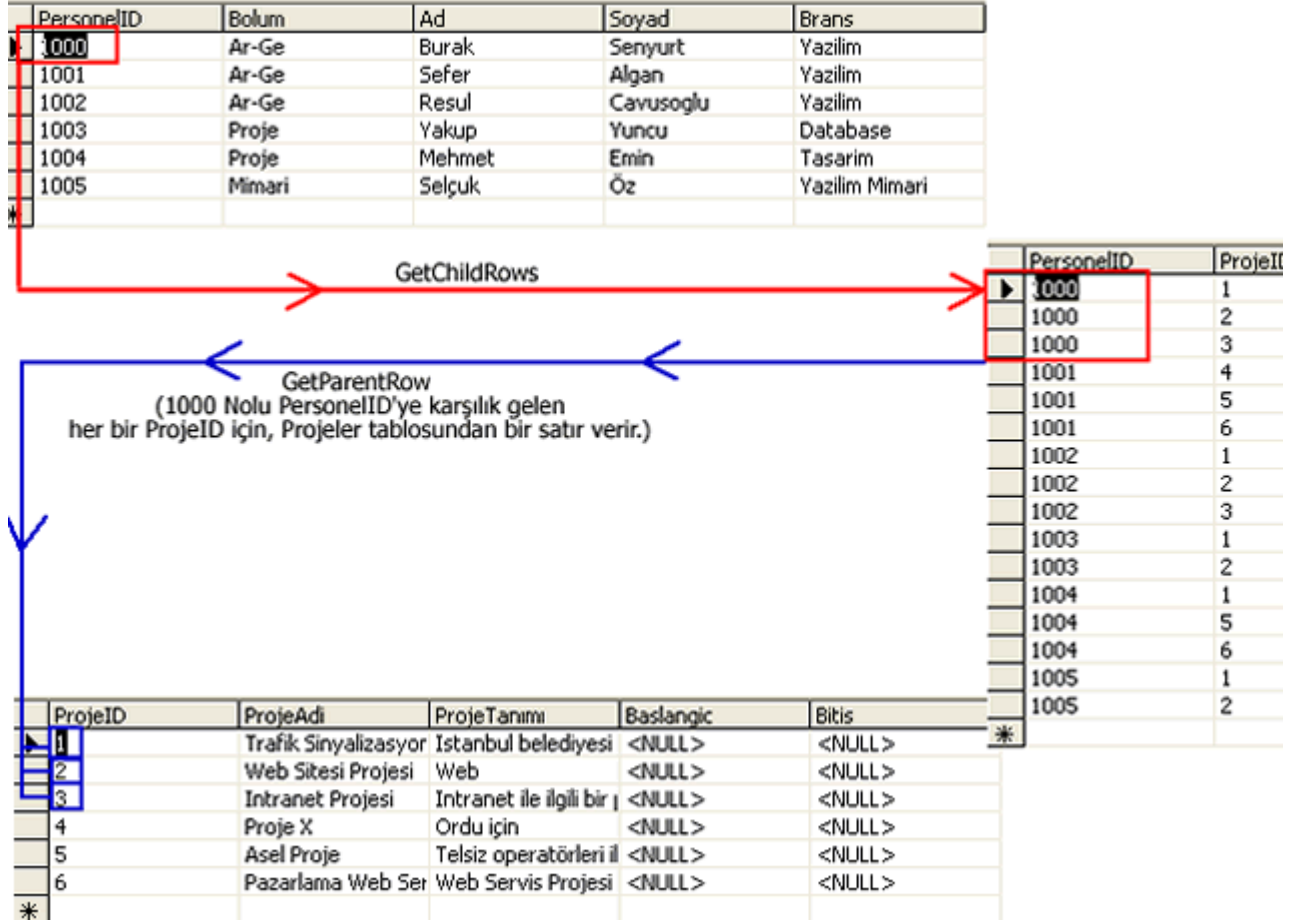
```
DataRow drProje=drProjeNo.GetParentRow("Projeler_MuhendisProje");
```

```
lbProjeler.Items.Add(drProje["ProjeAdi"]);
```

```
}
```

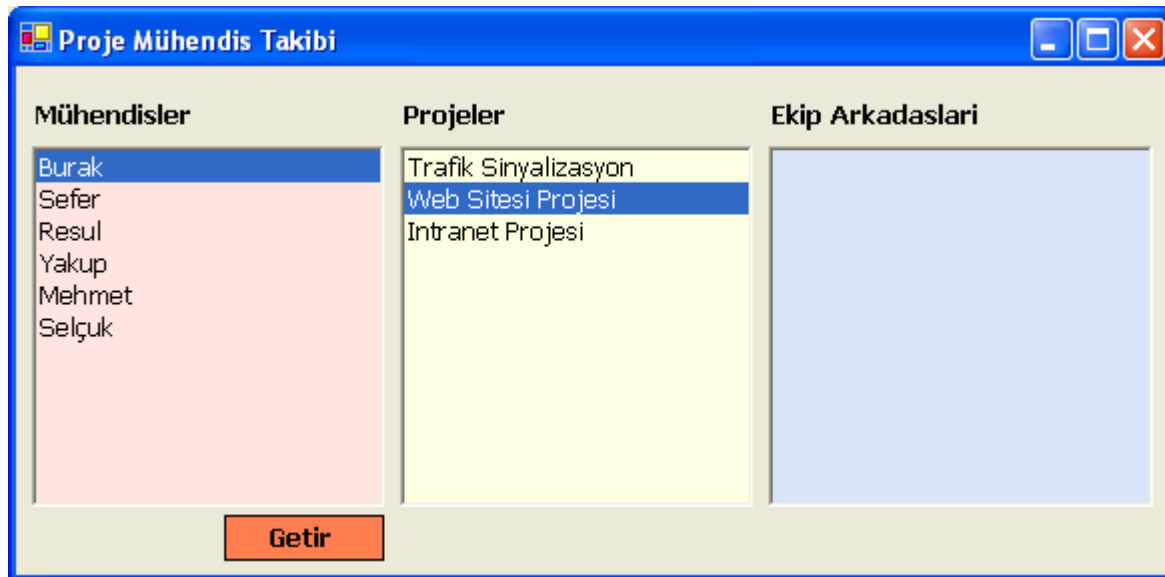
```
}
```

Kodumuzda neler olduğunu anlayabilmek için aşağıdaki şekil bize daha fazla yardımcı olacaktır. Burada, foreach döngüsü içerisinde meydana gelen olaylar tasvir edilmeye çalışılmıştır.



Şekil 7. Bir Mühendisin üzerinde çalıştığı projelerin elde edilmesi.

Uygulamayı çalıştırıp herhangi bir Mühendis için Getir başlıklı butona tıkladığımızda, bu mühendisin çalıştığı projelerin elde edildiğini görürüz.



Şekil 8. Uygulamanın çalışmasının sonucu.

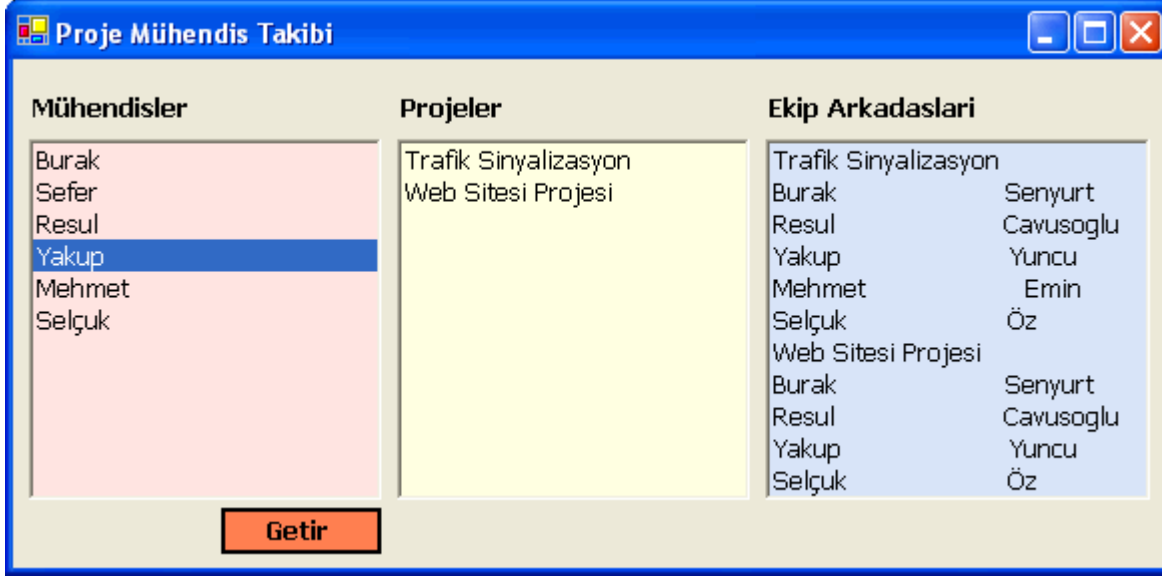
Şimdi gelelim ikinci kısma. Foreach döngüsü içinde, seçilen Mühendis satırının, MuhendisPersonel tablosundaki alanlarını GetChildRows metodu ile elde ettik. Daha sonra elde edilen her satırın, Projeler tablosunda karşılık geldiği satırlara ulaştık. Bu işlemi gerçekleştirmemizde, MuhendisProje tablosunun ve bu tablo ile Muhendis ve Projeler tablolarının aralarındaki bire-çok ilişkilerin büyük bir önemi vardır. Şimdi ise amacımız elde edilen projelere kimlerin katıldığı. İşte bunun için, elde edilen her proje satırından geriye doğru gidecek ve yine ilişkileri kullanarak bu problemi sonuca kavuşturacağız. Bunun için btnGetir olay prosedüründeki kodları aşağıdaki şekilde değiştirmemiz yeterlidir.

```
private void btnGetir_Click(object sender, System.EventArgs e)
{
    lbProjeler.Items.Clear();
    lbEkip.Items.Clear();

    DataRow dr;
    dr=dtMuhendisler.Rows.Find(lbMuhendisler.SelectedValue);

    foreach(DataRow drProjeNo in dr.GetChildRows("Muhendis_MuhendisProje"))
    {
        DataRow drProje=drProjeNo.GetParentRow("Projeler_MuhendisProje");
        lbProjeler.Items.Add(drProje["ProjeAdi"]);
        lbEkip.Items.Add(drProje["ProjeAdi"]);
        foreach(DataRow drMuh in drProje.GetChildRows("Projeler_MuhendisProje"))
        {
            DataRow drMuhBilgi=drMuh.GetParentRow("Muhendis_MuhendisProje");
            lbEkip.Items.Add(drMuhBilgi["Ad"]+" "+drMuhBilgi["Soyad"]);
        }
    }
}
```

Yeni düzenlemeler ile uygulamamızı çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 9. Bir Mühendisin çalıştığı projeler ve projelerdeki takım arkadaşlarının elde edilmesi.

Burada yaptıklarımız değerlendirecek kafa karıştırıcı tek unsurun foreach döngüsü içerisindeki yaklaşımlar olduğunu görürüz. Bunun yanında, ara tablomuz olan MuhendisProje tablosunun nasıl oluşturulduğu, verileri nasıl tuttuğu ve diğer tablolar arasındaki ilişkilerin .net ortamında bağlantısız katman üzerinde nasıl simüle edildiği önemlidir. Bu olgulara dikkat etmenizi ve iyice incelemenizi öneririm. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde buluşmak dileğiyle hepinize mutlu günler dilerim.

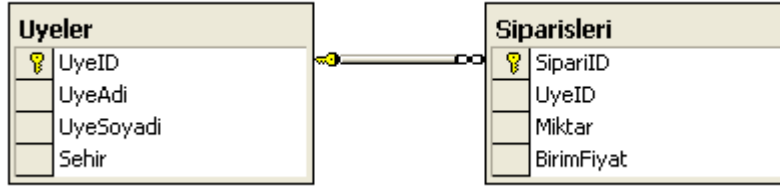
İlişkiler ve Hesaplanmış Alanların Bir Arada Kullanılması

Değerli Okurlarım, Merhabalar.

Bu makalemizde aralarında bire-çok (one-to-many) ilişki olan tablolar için hesaplanmış alanların, (yani DataColumn sınıfının Expression özelliği ile oluşturduğumuz sütunların) tablolar arasındaki ilişkiler ile nasıl bir arada kullanılabileceğini incelemeye çalışacağız. Burada bir arada kullanımdan kastım, örnek olarak; ebevyen (parent) tabloda fiziki olarak var olmayan ancak uygulamanın çalışması sırasında oluşturulacak bir sütundan, detay tablosundaki ilişkili alanlar üzerinden toplam, ortalama, miktar gibi Aggregate ifadelerinin çalıştırılmasından ve sonuçların yine parent tabloya yansıtılmasından bahsediyorum.

Konumuzun ana problemini daha iyi anlamak için şu örneği göz önünde bulunduralım. İnternet üzerinden ticaret yapan sitemizde kullanıcıların temel bilgileri ile, vermiş oldukları sipariş bilgilerinin ayrı iki tabloda tutulduğunu ve bu tablolar arasında bire-çok ilişki olduğunu varsayalım. Kendimize ait yönetici ekranlarında, her bir üye için, bu güne kadar vermiş olduğu siparişlerin toplam sayısını ve bu siparişlerin hepsine ödemiş olduğu toplam tutarları anlık olarak görmek istediğimizi varsayalım. Burada bize, ebevyen tabloda bir hesaplanmış alan gerekmektedir. Ancak hesaplanmış alan değerleri, detay tablosundaki veriler üzerinden gerçekleştirilmek zorundadır. İşte bu noktada devreye iki tablo arasında tanımlamış olduğumuz bire-çok ilişki girer.

Problemi ve ne yapmak istediğimizi kısaca anlattıktan sonra derseniz bunu gerçek bir uygulama üzerinde incelemeye başlayalım. Bu uygulamada, web sitesi üyeleri için tasarlanmış olan Üyeler tablosu ve bu üyelerin satın alım bilgilerini tutan Siparişler isimli tablolara bir windows uygulaması üzerinden erişmek istediğimizi varsayalım. Tablolarımızın sahip olacağı alanlar ve aralarındaki ilişki aşağıdaki şekilde yer almaktadır.



Şekil 1. Tablolarımızın Yapısı ve Aralarındaki Bire-Çok İlişki.

Bu iki tabloyu göz önüne aldığımızda, bir üyeye ait birden fazla siparişin olabileceğini görürüz. Uygulamamız bittiğinde, DataGrid nesnemizde, her bir üyenin bu güne kadar vermiş olduğu siparişlerin toplam sayısını ve ödemiş olduğu toplam miktarları gösterecek iki yeni sütunumuz olucak. Hiç vakit kaybetmeden uygulamamızın kodlarını yazmaya başlayalım. Önce, aşağıdaki gibi bir Form tasarlayalım.

Şekil 2. Form Tasarımımız.

Uygulamamızın kodlarına gelince.

```
SqlConnection con;
SqlDataAdapter da;
DataTable dtUyeler;
DataTable dtSiparisleri;
DataSet ds;
```

```
private void btnGetir_Click(object sender, System.EventArgs e)
{
    /* Sql sunucumuza olan bağlantımız oluşturuluyor. */
    con=new SqlConnection("data source=localhost;initial catalog=Friends;integrated security=SSPI");
```

```
ds=new DataSet(); /* DataSet nesnemiz oluşturuluyor. */
```

```
/* Önce, Uyeler tablomuzdaki verileri alıyor , dtUyeler DataTable'ına yüklüyor ve oluşan veri kümesini temsil eden bu DataTable nesnesinide DataSet nesnemizin tables koleksiyonuna ekliyoruz.*/
```

```
da=new SqlDataAdapter("Select * From Uyeler",con);
dtUyeler=new DataTable();
da.Fill(dtUyeler);
ds.Tables.Add(dtUyeler);
```

```
/* Aynı işlemi Siparisleri tablosu için yapıyoruz.*/
```

```
da=new SqlDataAdapter("Select * From Siparisleri",con);
dtSiparisleri=new DataTable();
da.Fill(dtSiparisleri);
ds.Tables.Add(dtSiparisleri);
```

```
/* Uyeler tablosundan Siparisleri tablosuna olan (dolayısıyla dtUyeler DataTable nesnesinin bellekte işaret ettiği bölgedeki veri satırlarından, dtSiparisleri dataTable nesnesinin bellekte temsil ettiği bölgedeki veri kümesine olan) bire-çok ilişkiyi tanımlıyoruz. */
```

```
ds.Relations.Add("Uyeler_Siparisleri",dtUyeler.Columns["UyeID"],dtSiparisleri.Columns["UyeID"],false);
```

```
dtSiparisleri.Columns.Add("ToplamTutar",typeof(Decimal),"Miktar*BirimFiyat");/* Bu satır ile, dtSiparisleri tablomuzda, her bir satır için Miktar ve BirimFiyat alanlarının değerlerini çarpıyoruz. Çıkan sonuçları ToplamTutar isimli yeni tanımladığımız bir alana içinde tutacak şekilde dtSiparisler DataTable nesnesinin Columns koleksiyonuna ekliyoruz. */
```

```
dtUyeler.Columns.Add("Siparis Sayisi",typeof(int),"COUNT(Child.UyeID)");/* Burada ise, dtUyeler tablosunda Siparis Sayisi isimli yeni bir alan oluşturuyoruz. Bu alan, ilişkili tablo olan detay tablosundaki UyeId alanlarının sayısını COUNT ile hesaplıyor. Ancak bunu yaparken Child nesnesini kullanıyor. Nitekim burada Child nesnesi, Uyeler tablosundaki her bir uyenin, Siparisleri tablosunda karşılık gelen satırlarını temsil ediyor. */
```

```
dtUyeler.Columns.Add("Toplam Ödeme",typeof(Decimal),"SUM(Child.ToplamTutar)");/* Burada ise, Child nesnesini kullanarak, var olan ilişki üzerinden, Siparisleri tablosuna gidiyor ve her bir üye için, az önce hesapladığımız ToplamTutar alanlarının toplamını SUM aggregate fonksiyonu ile hesaplıyoruz. Sonuçlarını ise, Toplam Ödeme isimli yeni bir alan olarak Uyeler tablomuzda ekliyoruz.*/
```

```
dgUyeler.DataSource=ds.Tables[0];
}
```

Uygulamamızda Child isimli nesneyi nasıl kullandığımıza ve bu sayede, iki tablo arasındaki ilişki yardımıyla, child tablodaki veriler üzerindeki hesaplamaları bir bütün halinde, parent tabloya hesaplanmış alan olarak nasıl eklediğimize lütfen dikkat ediniz. Uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ederiz. Gördüğünüz gibi her bir üye için, yapılmış olan toplam siparis sayısına ve ödemiş oldukları toplam miktarlara ulaşabildik.



ÜyelD	ÜyeAdı	ÜyeSoyadı	Şehir	Sipariş Sayısı	Toplam Ödeme
1	Burak	SENYURT	İstanbul	4	211
2	Sefer	ALGAN	İstanbul	2	252
3	Ahmet	NACAROGLU	İstanbul	2	260
4	Okan	BARLAS	İstanbul	2	1050

Getir

Şekil 3. Uygulamanın Çalışmasının Sonucu.

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Private Assembly ve Shared Assembly Kavramı

Değerli Okurlarım, Merhabalar.

Bu makalemizde, .NET'in temellerinden olan Assembly kavramının önemli bir bölümü olan Global Assembly Cache'i incelemeye çalışacağız. Net dilinde, assembly'ları private (özel) ve shared (paylaşımlı) olmak üzere iki kategoriye ayırabiliriz. Private assembly'lar oluşturulduklarında, çalıştırılabilmeleri için, uygulama ile aynı klasör altında yer almalıdırlar. Söz gelimi aşağıdaki gibi bir assembly'a sahip olduğumuzu düşünelim.

```
using System;
```

```
namespace PriAsm
{
    public class Hesap
    {
        public double Topla(double a,double b)
        {
            return a+b;
        }
    }
}
```

Visual Studio.Net ortamında bir class library olarak oluşturduğumuz bu assembly derlendiğinde, PriAsm.dll dosyasının oluşturulduğunu görürüz.



Şekil 1. Assembly dosyamız.

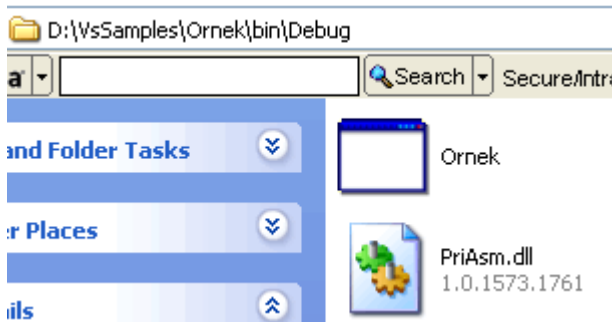
Şimdi, oluşturulan bu assembly içindeki Hesap isimli sınıfımızı, başka bir klasörde yer alan bir uygulama içerisinde kullanmak istediğimizi düşünelim. Bu durumda, PriAsm.dll dosyamızı, kullanmak istediğimiz uygulamanın klasörüne kopyalamamız yeterli olacaktır. Örneğin aşağıdaki uygulamayı göz önüne alalım.

```
using System;
using PriAsm;

namespace Ornek
{
    class Class1
    {
        static void Main(string[] args)
        {

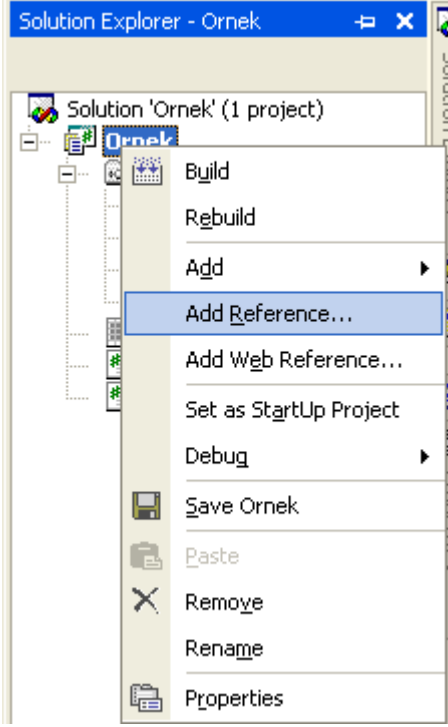
        }
    }
}
```

Bu Console uygulamasını derlemeye çalıştığımızda, **"The type or namespace name 'PriAsm' could not be found (are you missing a using directive or an assembly reference?)"** hatasını alırsınız. Bu hatayı almamız son derece doğaldır. Çünkü PriAsm.dll assembly'ımız private bir assembly'dir. Bunun doğal sonucu uygulamamızın bu assembly hakkında hiç bir bilgiye sahip olmamasıdır. Uygulamamıza, PriAsm.dll assembly'nin referans edilmesi gerekmektedir. Öncelikle, PriAsm.dll dosyasını, uygulamamızın assembly'nin bulunduğu klasöre kopylamamız gerekir.



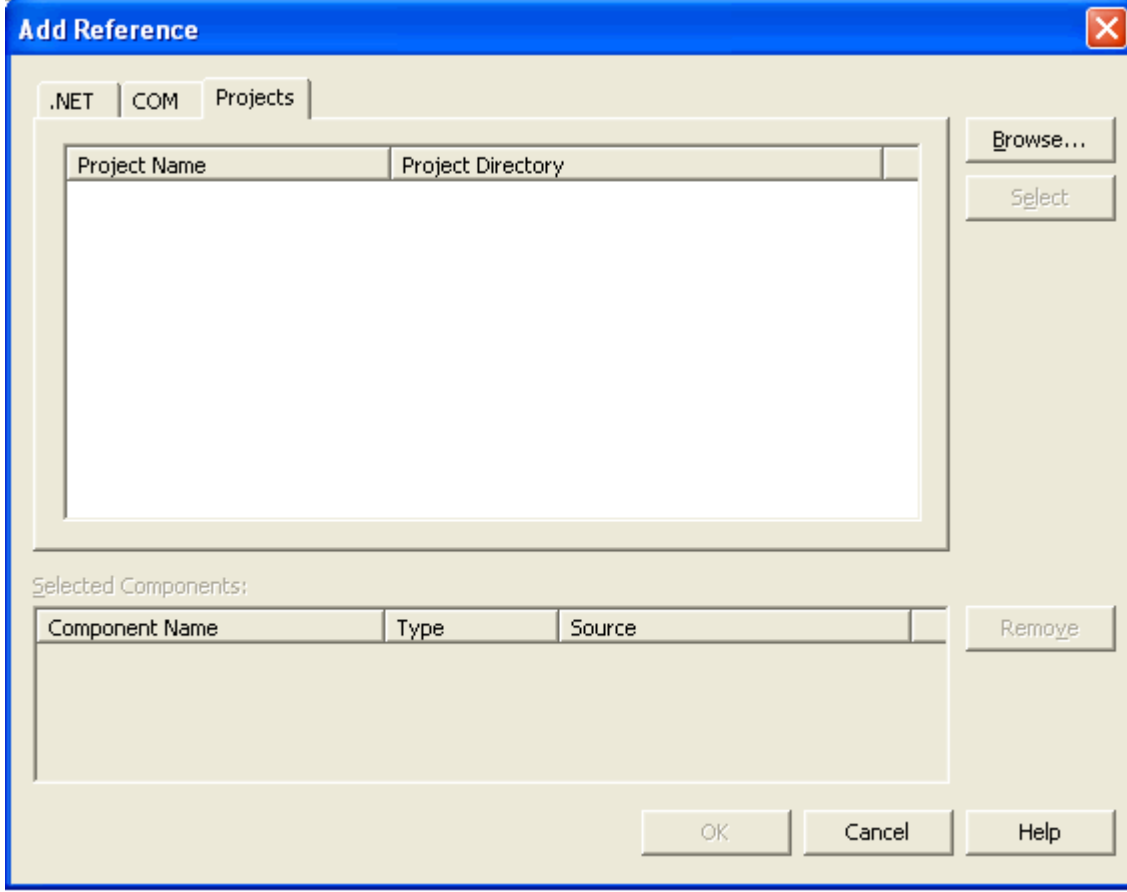
Şekil 2. PriAsm.dll ile uygulama assemble'i aynı yerde olmalı.

Daha sonra ise uygulamamıza PriAsm.dll assembly'ini referans etmemiz gerekir. Bu amaçla solution explorer'da assembly ismine sağ tıklanır ve Add Reference seçilir.



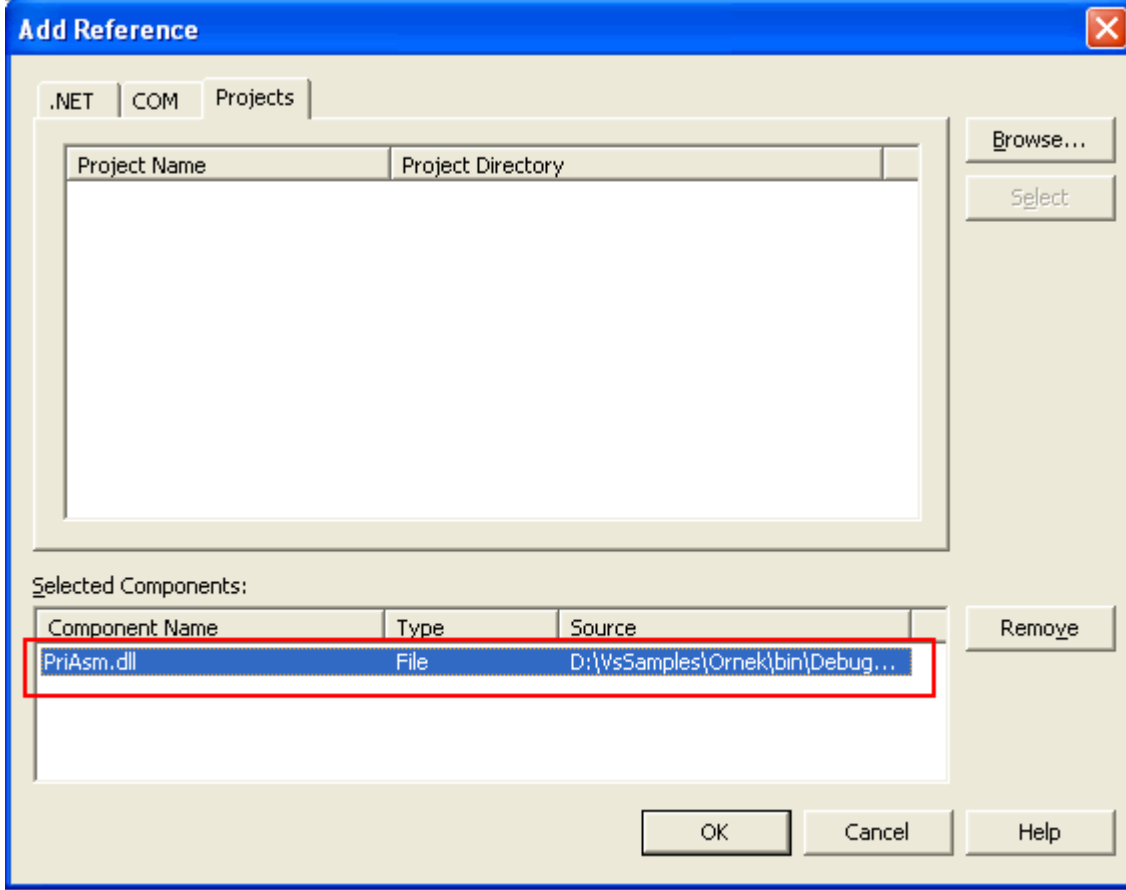
Şekil 3. Add Reference.

Karşımıza gelen Add Reference penceresinden, Projects kısmına geçilir.



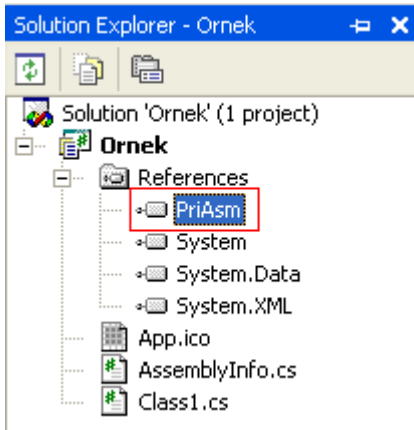
Şekil 4. Projects

Burada Browse seçeneği ile, PriAsm.dll assembly'nin bulunduğu klasöre gidilir ve bu dll dosyası seçilerek referans ekleme işlemi tamamlanmış olur.



Şekil 5. Referansın eklenmesi.

Bu noktadan sonra Solution Explorer penceresine baktığımızda, PriAsm isim alanının eklenmiş olduğunu görürüz.



Şekil 6. PriAsm eklenmiş durumda.

Artık uygulamamızda bu assembly içindeki sınıfları kolayca kullanabiliriz. Aşağıdaki basit örnekte bunu görebiliriz.

```
using System;
using PriAsm;
```

```
namespace Ornek
```

```
{
    class Class1
    {
        static void Main(string[] args)
        {
            Hesap h=new Hesap();
            double sonuc=h.Topla(4,5);
            Console.WriteLine(sonuc.ToString());
        }
    }
}
```

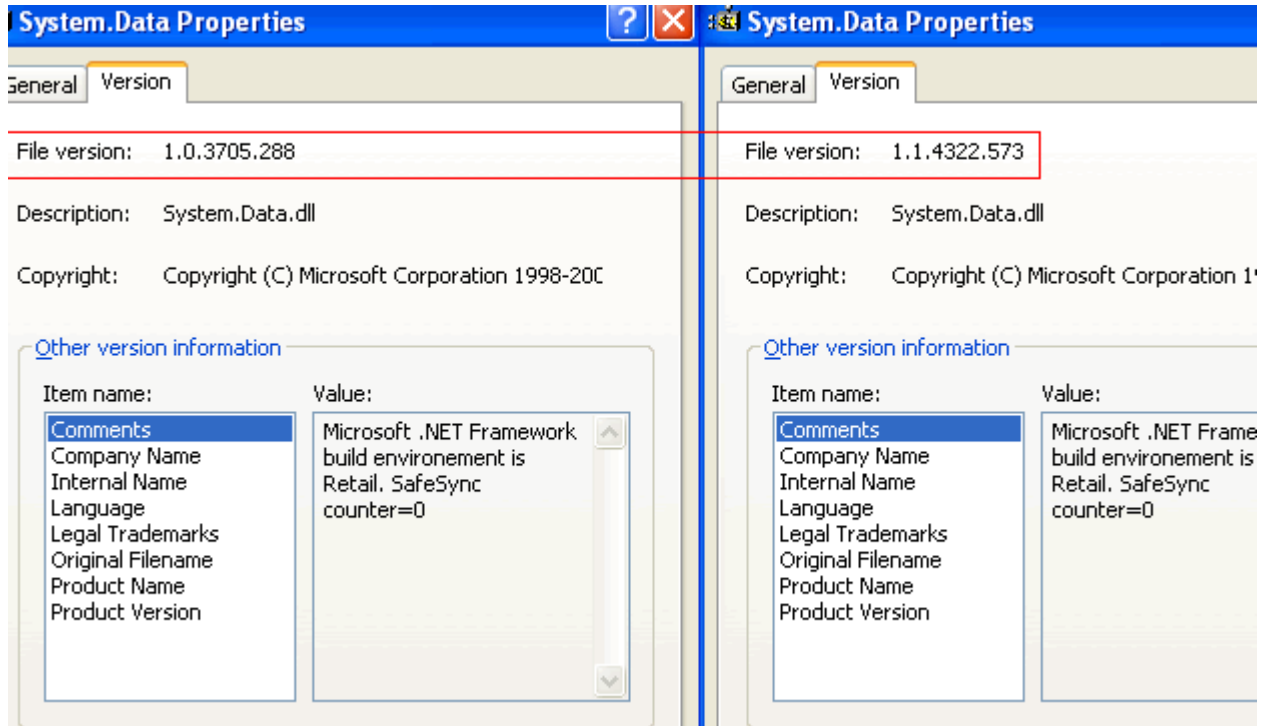
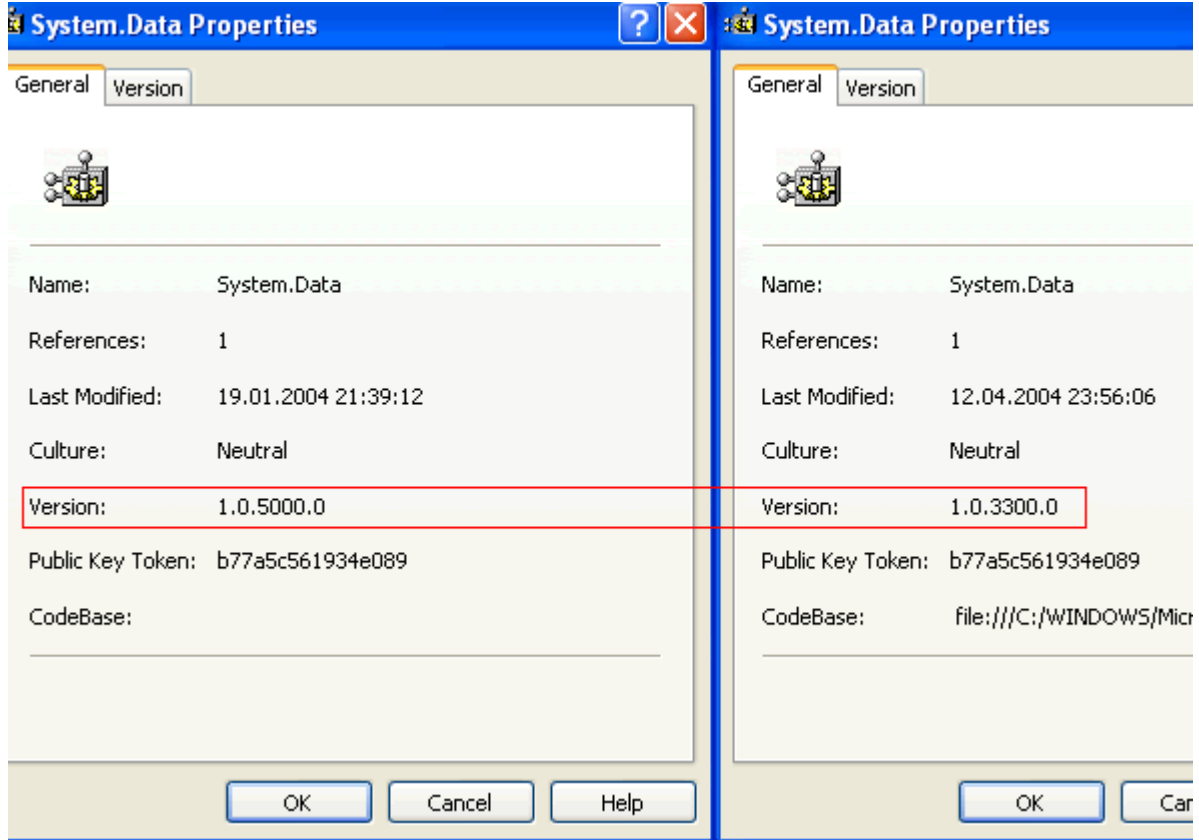
Private assembly'ları bu şekilde kullanmak her zaman tercih edilen bir yol değildir. Nitekim, PriAsm.dll assembly'ının erişilebilirlik bu dll'in, uygulamanın assembly'ı ile aynı klasörde olması gerekmektedir. (Nitekim, PriAsm.dll assembly'ının uygulamanın debug klasöründen başka bir yere taşınması, uygulamanın derleme zamanında hata vermesine yol açar. Çünkü belirtilen adresteki referans dosyası yerinde değildir.) Buna karşılık olarak bazen, oluşturduğumuz assembly'a birden fazla uygulamanın aynı yerden erişmesini isteyebiliriz. İşte bu durumda devreye Global Assembly Cache girmektedir. GAC .NET uygulamaları tarafından paylaşılan bileşenlerin yer aldığı bir veri deposudur. Birden fazla uygulamanın ortak olarak kullanacağı assembly'lar sistemdeki Global Assembly Cache 'e yüklenerek paylaşılmış(shared) assembly'lar oluşturabiliriz. GAC 'da tutulan assembly'ların görüntüsüne, bir Win XP sisteminde C:\WINDOWS\assembly klasöründen ulaşabiliriz. Burada yer alan assembly'lar, C# kodu ilk olarak yürütüldüğünde anında derlenir ve GAC önbelleğinde tutulurlar.

Global Assembly Name	Type	Version	Culture	Public Key Token
System.CF.Windows.Forms.DataGrid		7.0.5000.0		b03f5f7f11d50a3a
System.Configuration.Install		1.0.5000.0		b03f5f7f11d50a3a
System.Configuration.Install		1.0.3300.0		b03f5f7f11d50a3a
System.Data		1.0.5000.0		b77a5c561934e089
System.Data		1.0.3300.0		b77a5c561934e089
System.Data.OracleClient		1.0.5000.0		b77a5c561934e089
System.Design	Native Images	1.0.5000.0		b03f5f7f11d50a3a
System.Design	Native Images	1.0.5000.0		b03f5f7f11d50a3a
System.Design		1.0.5000.0		b03f5f7f11d50a3a
System.Design		1.0.3300.0		b03f5f7f11d50a3a
System.DirectoryServices		1.0.5000.0		b03f5f7f11d50a3a
System.DirectoryServices		1.0.3300.0		b03f5f7f11d50a3a
System.Drawing	Native Images	1.0.5000.0		b03f5f7f11d50a3a
System.Drawing	Native Images	1.0.5000.0		b03f5f7f11d50a3a

Şekil 7. GAC Assembly'ları.

Burada dikkat edecek olursanız örneğin, System.Data Assembly'ından iki adet bulunmaktadır. Bu iki assembly'in sistemde sorunsuz bir şekilde çalışması, assembly'ların kimliklerinin farklılığı sayesinde mümkün olmaktadır. Bu farklılığı yaratan GAC içine kurulan her bir assembly'in farklı strong name'lere sahip olmasıdır. Bir strong name, bir assembly'in adı, versiyon numarası, dil bilgileri, dijital

imzaları ve public anahtar değeri bilgilerinden oluşur. Örneğin System.Data assembly'nin iki versiyonu arasındaki farklar aşağıdaki şekilde görüldüğü gibidir.



Şekil 8. Farklılıklar.

Bir assembly'ın yukarıda bahsedilen bilgileri AssemblyInfo isimli dosyada tutulmaktadır.Şimdi dilerseviz geliştirmiş olduğumuz PriAsm.dll assembly'ını GAC'e nasıl kayıt edeceğimizi incelemeye çalışalım. İlk olarak bize bu assembly'ı sistem için benzersiz (unique) yapacak bir strong name gerekli. Bir strong name üretmek için, .Net Framework'un sn.exe tool'unu kullanabiliriz.

```

C:\ Command Prompt
D:\UsSamples\PriAsm\bin\Debug>sn -k Anahtar.sif
Microsoft (R) .NET Framework Strong Name Utility Version 1.0.3705.0
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.
Key pair written to Anahtar.sif
D:\UsSamples\PriAsm\bin\Debug>

```

Şekil 9. Strong Name'in oluşturulması.

Bu işlemin ardından oluşan Anahtar.sif isimli dosyayı, assembly'ımıza bildirmemiz gerekiyor. Bunun için AssemblyInfo dosyasındaki AssemblyKeyFile niteliğini kullanacağız. (Burada dosya uzantısının sif olmasının özel bir nedeni yok. Ben sifre'nin sif'ini kullandım. Sonuç olarak dosya binary yazılacağı için herhangi bir format verilebilir.)

```
[assembly: AssemblyKeyFile("D:\\vssamples\\PriAsm\\bin\\debug\\Anahtar.sif")]
```

Artık assembly'ımız bir strong name'e sahip. Dolayısıyla GAC içindeki assembly'lardan ve sonradan gelabilecek olan assembly'lardan tamamıyla farklı bir yapıya büründü. Artık oluşturduğumuz bu assembly'ı GAC'e alabiliriz. Bunu iki yolla gerçekleştirebiliriz. İlk olarak, .Net Framework'ün GACUtil.exe tool'unu bu iş için kullanabiliriz.

```

C:\ Command Prompt
D:\UsSamples\PriAsm\bin\Debug>gacutil /i PriAsm.dll
Microsoft (R) .NET Global Assembly Cache Utility. Version 1.0.3705.0
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.
Assembly successfully added to the cache
D:\UsSamples\PriAsm\bin\Debug>

```

Şekil 10. Assembly'ın GAC'e kurulması.

Bu işlemin ardından GAC klasörüne baktığımızda, PriAsm assembly'ımızın eklenmiş olduğunu görürüz.

Office	11.0.0.0	71e9bce111e9429c
PriAsm	1.0.1573.3422	52ac6d32da4139ee
DenCode	1.0.5000.0	b03f5f7f11d50a3a

Şekil 11. Assembly'ın GAC'e eklenmesi.

Diğer yandan aynı işlemi, PriAsm.dll dosyasını bu klasöre sürükleyerekte gerçekleştirebiliriz. Artık bu noktadan itibaren, PriAsm assembly'ına ve içindeki sınıflara, herhangi bir .net uygulamasından kolayca erişebiliriz. Örneğin, herhangi bir .net uygulamasından PriAsm assembly'ına ulaşmak için tek yapmamız gereken assembly'ı uygulamamıza aynı private assembly'larda olduğu gibi referans etmektir. Fakat bu sefer, PriAsm.dll assembly'ını uygulamamızın PE(Portable Executable) dosyasının bulunduğu debug klasörüne almak gibi bir zorunluluğumuz yoktur. Çünkü, programın ilk derlenişinde, PriAsm.dll, GAC'e alınır ve burada tutulur. Dolayısıyla aşağıdaki örnek kodların yer aldığı Console Uygulamasının bulunduğu debug klasörüne PriAsm.dll dosyasının yüklenmesi gerekmez.

```
using System;
using PriAsm;
namespace ConsoleApplication3
{
    class Class1
    {
        static void Main(string[] args)
        {
            Hesap h=new Hesap();
            double sonuc=h.Topla(1,2);
        }
    }
}
```

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Single File Assembly ve Multiple-File Assembly Kavramları

Değerli Okurlarım, Merhabalar.

Bir önceki makalemizde, assembly'ları erişilebilirliklerine göre özel (private) ve paylaşılmış (shared) olmak üzere iki kategoriye ayırabileceğimizi incelemiştik. Assembly'ları ayrıca, tek dosya (single file) ve çoklu dosya (multiple-file) olmak üzere iki farklı kategoriye daha ayırabiliriz. Bu makalemizde assembly'ların bu tiplerini incelemeye çalışacağız.

Çoğunlukla .net ile uygulama geliştirirken, gerçekleştirmiş olduğumuz uygulamalar tek bir assembly' dan oluşurlar. Bu varsayılan olarak böyledir. İşte bu assembly' lar single file (tek dosya) assembly olarak adlandırılır. Bir multiple-file assembly uygulaması ise, birden fazla assembly'dan oluşabilir. Öyleki tüm bu assembly'lar farklı .net dillerince yazılmış uygulama parçaları olabilir. Tüm bunların bir araya getirilerek PE(portable executable) olarak kullanılacak teki bir assembly altında birleştirilmesi ile, multiple-file assembly mimarisi elde edilmiş olur.

Örneğin, vb.net module'lerinden, J# kütüphanelerinden oluşan bir uygulamada, giriş noktasına sahip olan uygulamanın (yani Main metodunu içeren uygulamanın) C# ile yazılmış olduğunu ve bu dosyadan diğer module ve kütüphanelere ait elemanların kullanıldığını düşünelim. Burada PE(portable executable)'a ait manifesto bilgisi, giriş noktasına sahip olan assembly için oluşturulacaktır. Bir assembly manifesto' su, uygulamanın başvurduğu diğer assembly'lara ait bilgileri, assembly'da kullanılan tür bilgilerini, assembly'a ait dosyalara ait bilgileri, izin bilgilerini vb. içerir. Dolayısıyla multiple-file assembly' a ait manifesto içerisinde, multiple-file assembly' ı oluşturan diğer module'lere, kütüphanelere ait bilgilerde yer alacaktır. Böylece yalın bir tabirle elimizde, bir

kaç .net dili ile yazılmış parçalardan oluşan ve tek bir assembly altında birleştirilen bir uygulama olacaktır. Bu bizim, farklı .net dilleri ile yazılmış uygulama parçalarını tek bir assembly altında birleştirerek kullanabilmemizi ve uygulamamızı yapılandırabilmemizi sağlar.

Örnekler üzerinden gittiğimizde bu konuyu daha iyi kavrayacağınızı düşünüyorum. İlk olarak çok basit bir single-file assembly oluşturacak ve yapısını, .net araçlarından ildasm (Intermediate Language DisAssembly) ile inceleyeceğiz. Aşağıda C# dili ile yazılmış basit bir assembly görüyorsunuz.

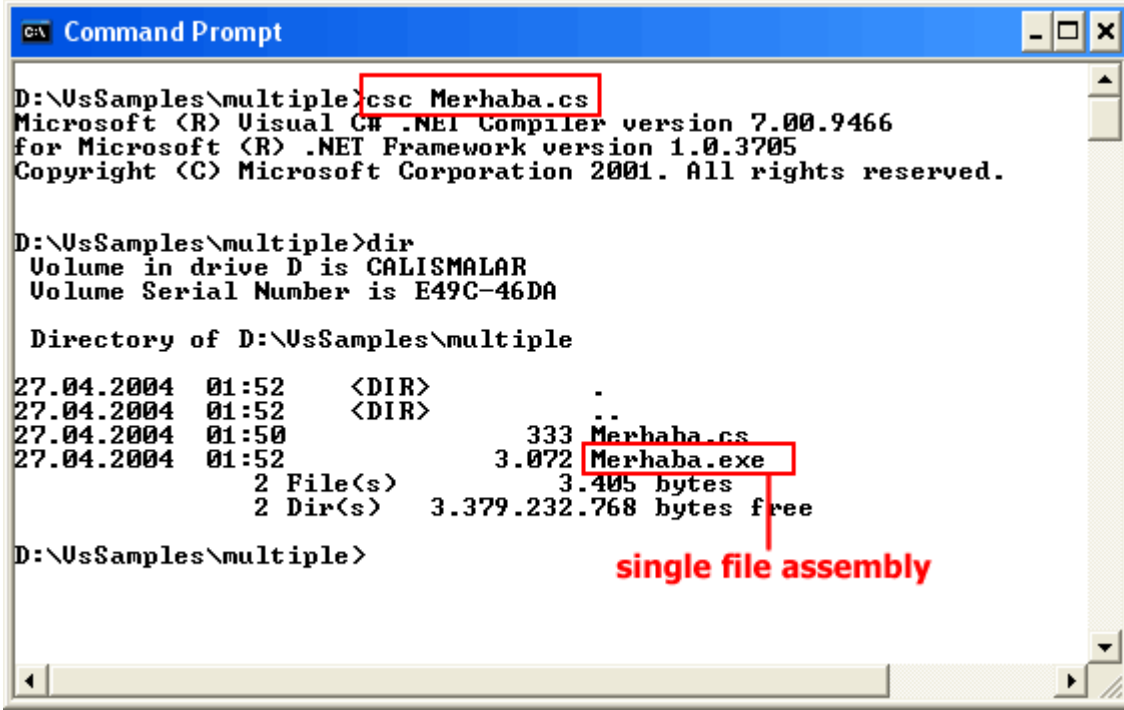
```
using System;

public class Giriş
{
    public static void Main(string[] args)
    {
        int a,b;
        a=5;
        b=6;
        int sonuc=Topla(a,b);

        Console.WriteLine("Single File Assembly");
        Console.WriteLine("Toplam {0}",sonuc.ToString());
    }

    public static int Topla(int birinci,int ikinci)
    {
        return birinci+ikinci;
    }
}
```

Bu kodu aşağıdaki şekilde görüldüğü gibi derlediğimizde, Merhaba.exe assembly'ının oluşturulduğunu görürüz. Bu assembly'ımız bir Main metodu içerdiğinden ve biz bu assembly'ı exe olarak derleyip ortaya bir PE çıkarttığımızdan,bu assemble'a ait manifesto bilgileri de buna göre olacaktır.



Şekil 1. Single Assembly

Şimdi ildasm aracını kullanarak, oluşturmuş olduğumuz bu assembly'ın içeriğine bakalım.



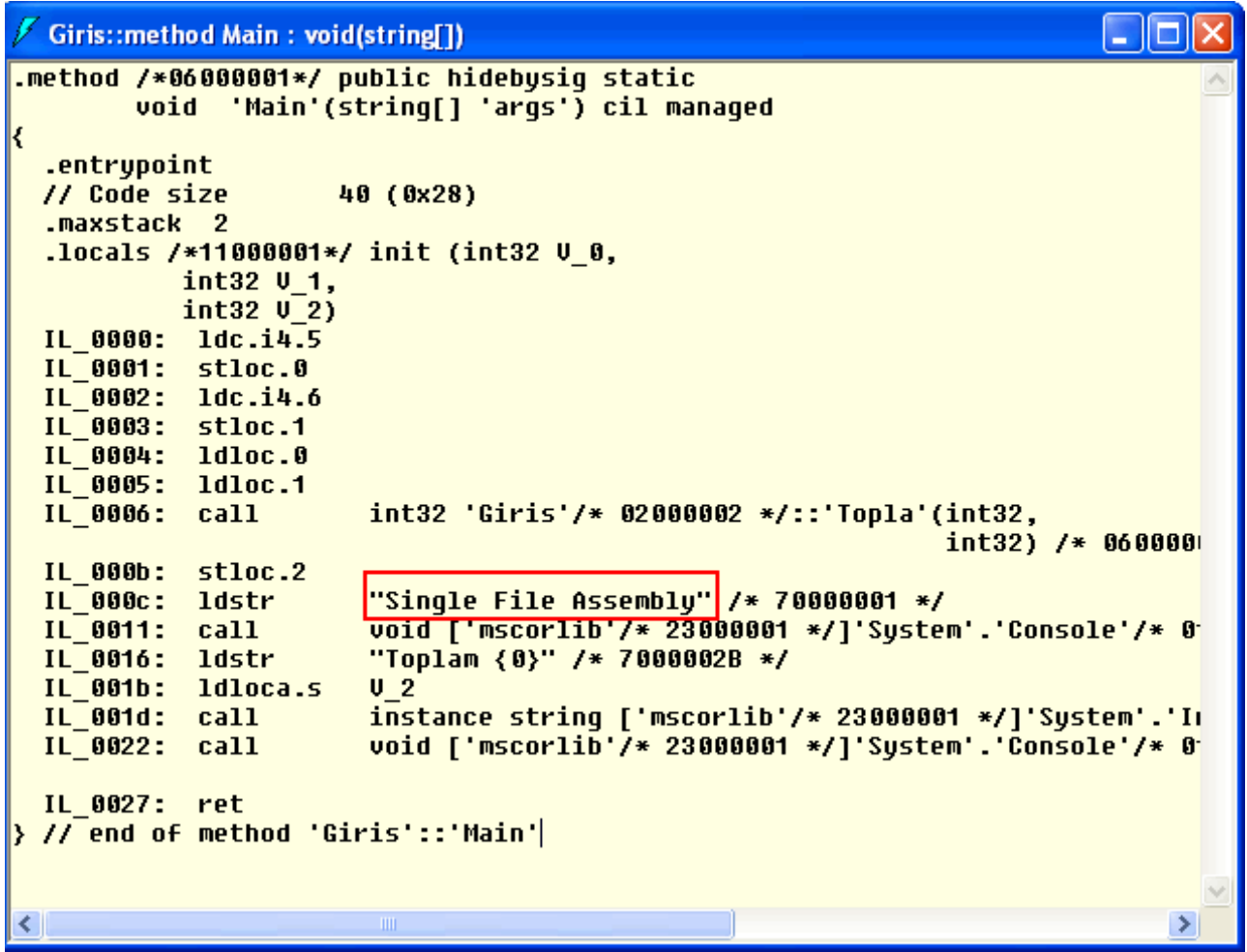
Şekil 2. Giriş.exe assembly'nin içeriği.

Burada görüldüğü gibi, assembly'ımıza ait manifesto bilgisinde PE olduğuna dair en büyük işareti gösteren entry point bilgisinin yer aldığı Main metodu görülmektedir. Daha önceden belirttiğimiz gibi bu manifesto içerisinde, Merhaba.exe assembly'nin kullandığı başka assembly'lara ait referans bilgileri, assembly'a ait module bilgisi, hashcode bilgileri vb. bulunur. Tüm bu manifesto bilgileri

metadata verisi olarak Merhaba.exe assembly'ı içerisinde tutulmaktadır. Bu assembly bir PE(portable executable) olduğu için single assembly olarak değerlendirilir. Merhaba.exe assembly'ımızın manifesto bilgileri aşağıdaki gibidir.

```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
  .ver 1:0:3300:0
}
.assembly Merhaba
{
  // --- The following custom attribute is added automatically, do not uncomment -----
  // .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute::.ctor(bool,
  // bool) = ( 01 00 00 01 00 00 )
  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.module Merhaba.exe
// MVID: {C9BE521B-50BF-49DE-A9F9-F6EDB3D31941}
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 512
.corflags 0x00000001
// Image base: 0x07000000
```

Ancak bu assembly'ın bir single-file assembly olduğunun en büyük kanıtı Main metoduna ait IL kodlarında yazmaktadır.



```

Giris::method Main : void(string[])
.method /*06000001*/ public hidebysig static
    void 'Main'(string[] 'args') cil managed
{
    .entrypoint
    // Code size          40 (0x28)
    .maxstack 2
    .locals /*11000001*/ init (int32 V_0,
        int32 V_1,
        int32 V_2)
    IL_0000: ldc.i4.5
    IL_0001: stloc.0
    IL_0002: ldc.i4.6
    IL_0003: stloc.1
    IL_0004: ldloc.0
    IL_0005: ldloc.1
    IL_0006: call        int32 'Giris'/* 02000002 */::'Topla'(int32,
        int32) /* 06000001 */
    IL_000b: stloc.2
    IL_000c: ldstr        "Single File Assembly" /* 70000001 */
    IL_0011: call        void ['mscorlib'/* 23000001 */]'System'. 'Console'/* 0
    IL_0016: ldstr        "Toplam {0}" /* 7000002B */
    IL_001b: ldloc.a.s    V_2
    IL_001d: call        instance string ['mscorlib'/* 23000001 */]'System'. 'I
    IL_0022: call        void ['mscorlib'/* 23000001 */]'System'. 'Console'/* 0
    IL_0027: ret
} // end of method 'Giris'::'Main'

```

Şekil 3. Single-File Assembly Kanıtı.

Her .net assembly'ında olduğu gibi burada da manifestomuz, standart olarak, mscorlib kütüphanesine bir başvuru ile başlamaktadır. Daha sonra assembly'a ait bilgiler başlar. Bir single assembly bu manifesto bilgileri dışında elbette uygulama kodlarının MSIL karşılıklarında içermektedir. Bir single file assembly'ın temel yapısı aşağıdaki gibidir.

Assembly



Şekil 4. Single-File Assembly'ların genel yapısı.

Gelelim, multiple-file assembly'lara. Bu assembly türünün anlamının en iyi yolu konuyu bir senaryo üzerinde düşünmektir. Uygulamamızın, vb.net ile yazılmış bir module'ü, J# ile yazılmış bir kütüphanesi olduğunu farzedelim. Biz bu kaynaklardaki sınıfları asıl uygulamamızda yani PE olucak assembly'ımızda kullanmaya çalışacağız. Öncelikle işe, vb.net module'ünü oluşturmak ile başlayalım.

```
namespace vbdotnet
    public class vbse lam
        public shared sub Selam()
            System.Console.WriteLine("Selam, burası VB.NET module")
        end sub
    end class
end namespace
```

Şimdi bu vb.net kod dosyasını bir netmodule dosyası olarak derleyeceğiz. Bunun için aşağıdaki tekniği uygulayacağız.

```

C:\ Command Prompt

D:\UsSamples\multiple>vbc /t:module vbse lam.vb
Microsoft (R) Visual Basic .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.00.3705.288
Copyright (C) Microsoft Corporation 1987-2001. All rights reserved.

D:\UsSamples\multiple>dir
Volume in drive D is CALISMALAR
Volume Serial Number is E49C-46DA

Directory of D:\UsSamples\multiple

27.04.2004 02:12 <DIR>      .
27.04.2004 02:12 <DIR>      ..
27.04.2004 02:09           333 Merhaba.cs
27.04.2004 01:52          3.072 Merhaba.exe
27.04.2004 02:12          2.048 vbse lam.netmodule
27.04.2004 02:12          171 vbse lam.vb
                4 File(s)          5.624 bytes
                2 Dir(s)    3.379.224.576 bytes free

D:\UsSamples\multiple>_
```

Şekil 5. vb.net module

Lütfen, net module dosyasını vbc derleyici aracı ile nasıl oluşturduğumuza dikkat edelim. Bu module, vb.net kodları ile yazılmış olup herhangi bir giriş noktası içermemektedir. Bu nedenle bu assembly aslında bir PE değildir. Tamamıyla başka bir PE assembly'ının kullanabilmesi için geliştirilmiştir. Bunu yazılım ekibinizin vb.net programcılarının geliştirmiş olduğu bir module olarak düşünebilirsiniz. Bizim amacımızda, bu module'ü asıl PE assembly'ımız içerisinde kullanmak ve bir multiple-assembly meydana getirmektir. Şimdide J# kodlarından oluşan bir kütüphane geliştirelim.


```

import System;

package Araclar
{
    public class jsharpselem
    {
        public static function Selam()
        {
            System.Console.WriteLine("Selam, burasi JSharp ekibi");
        }
    }
}

```

The screenshot shows a Windows Command Prompt window with the following text:

```

C:\> Command Prompt

D:\UsSamples\multiple>jsc /t:library jsharpselem.js
Microsoft (R) JScript .NET Compiler version 7.00.9502
for Microsoft (R) .NET Framework version 1.00.3705
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

D:\UsSamples\multiple>dir
Volume in drive D is CALISMALAR
Volume Serial Number is E49C-46DA

Directory of D:\UsSamples\multiple

27.04.2004  02:31    <DIR>          .
27.04.2004  02:31    <DIR>          ..
                4.096 jsharpselem.dll
                178 jsharpselem.js
                333 Merhaba.cs
27.04.2004  01:52    3.072 Merhaba.exe
27.04.2004  02:12    2.048 vbselem.netmodule
27.04.2004  02:12    171 vbselem.vb
                6 File(s)          9.898 bytes
                2 Dir(s)      3.379.216.384 bytes free

D:\UsSamples\multiple>

```

Şekil 6. J# ile yazılmış kütüphanemiz.

Sıra geldi tüm bu assembly'ları kullanacak olan assembly'ımızı yazmaya. Bu , C# ile yazılmış bir assembly olacak ve aynı zamanda, yukarıda yazmış olduğumuz vb.net module'ünü ve j# kütüphanesini kullanacak.

```

using System;
using Araclar;

public class Temel
{

```

```

public static void Main(string[] args)
{
    System.Console.WriteLine("Ana uygulama");
    jsharpselem.Selam();
    vbdotnet.vbselem.Selam();
}
}

```

Kodumuzda, vb.net module'ü içindeki Selam ve j# ile yazılmış kütüphane içindeki Selam metodlarına erişiyoruz. Şimdi yapmamız gereken işlem bu assembly'ı bir netmodule olarak derlemek ve ardından tüm assembly'lar tek bir assembly altında birleştirmek. PE olucak olan assembly'ımızı netmodule haline getirmek için aşağıdaki kod satırını kullanacağız.

```

C:\> Command Prompt

D:\UsSamples\multiple>csc /addmodule:vbselem.netmodule /r:jsharpselem.dll,Microsoft.JScript.dll /target:module Temel.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

D:\UsSamples\multiple>dir
Volume in drive D is CALISMALAR
Volume Serial Number is E49C-46DA

Directory of D:\UsSamples\multiple

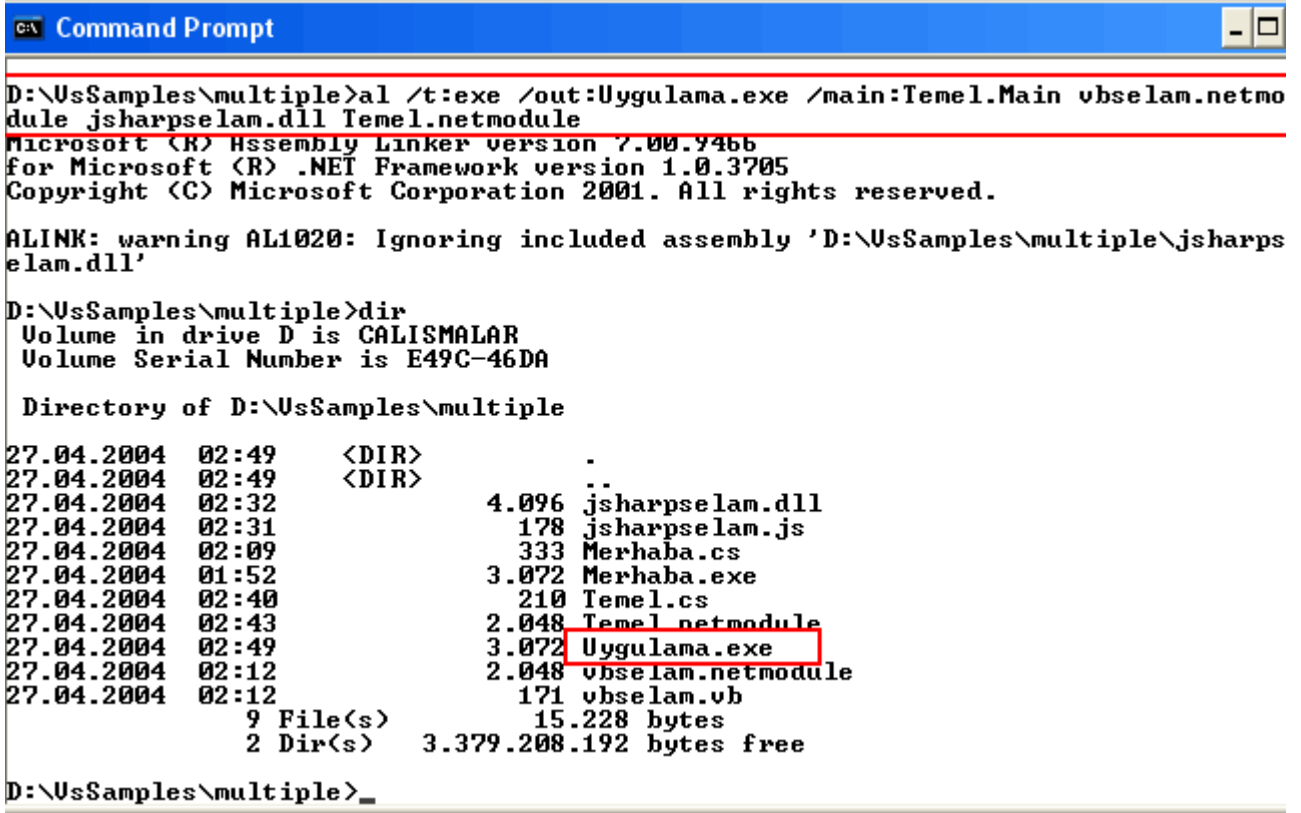
27.04.2004 02:41 <DIR> .
27.04.2004 02:41 <DIR> ..
27.04.2004 02:32 4.096 jsharpselem.dll
27.04.2004 02:31 178 jsharpselem.js
27.04.2004 02:09 333 Merhaba.cs
27.04.2004 01:52 3.072 Merhaba.exe
27.04.2004 02:40 210 Temel.cs
27.04.2004 02:43 2.048 Temel.netmodule
27.04.2004 02:12 2.048 vbselem.netmodule
27.04.2004 02:12 171 vbselem.vb
                8 File(s)      12.156 bytes
                2 Dir(s)    3.379.212.288 bytes free

D:\UsSamples\multiple>

```

Şekil 7. PE uygulamamızın netmodule olarak oluşturulması.

Bu komut satırında, PE olucak assembly'ımızı netmodule haline getirirken, kullandığımız j# kütüphanesini nasıl referans ettiğimize ayrıca vb.net module'ünü nasıl eklediğimize dikkat edelim. Sıra, tüm bu assembly'ları tek bir çatı altında toplayıp multiple-assembly'ımızı oluşturmaya geldi. Bu işi gerçekleştirmek amacıyla, AL (assembly linker) aracını kullanacağız. Bu araç yardımıyla, yazmış olduğumuz üç assembly'da tek bir assembly içinde toplayacak ve multiple-assembly yapımızı gerçekleştirmiş olacağız.



```

C:\ Command Prompt

D:\UsSamples\multiple>al /t:exe /out:Uygulama.exe /main:Temel.Main vbse lam.netmodule jsharpse lam.dll Temel.netmodule
Microsoft (R) Assembly Linker version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

ALINK: warning AL1020: Ignoring included assembly 'D:\UsSamples\multiple\jsharpse lam.dll'

D:\UsSamples\multiple>dir
Volume in drive D is CALISMALAR
Volume Serial Number is E49C-46DA

Directory of D:\UsSamples\multiple

27.04.2004 02:49 <DIR> .
27.04.2004 02:49 <DIR> ..
27.04.2004 02:32 4.096 jsharpse lam.dll
27.04.2004 02:31 178 jsharpse lam.js
27.04.2004 02:09 333 Merhaba.cs
27.04.2004 01:52 3.072 Merhaba.exe
27.04.2004 02:40 210 Temel.cs
27.04.2004 02:43 2.048 Temel.netmodule
27.04.2004 02:49 3.072 Uygulama.exe
27.04.2004 02:12 2.048 vbse lam.netmodule
27.04.2004 02:12 171 vbse lam.vb
          9 File(s)          15.228 bytes
          2 Dir(s)  3.379.208.192 bytes free

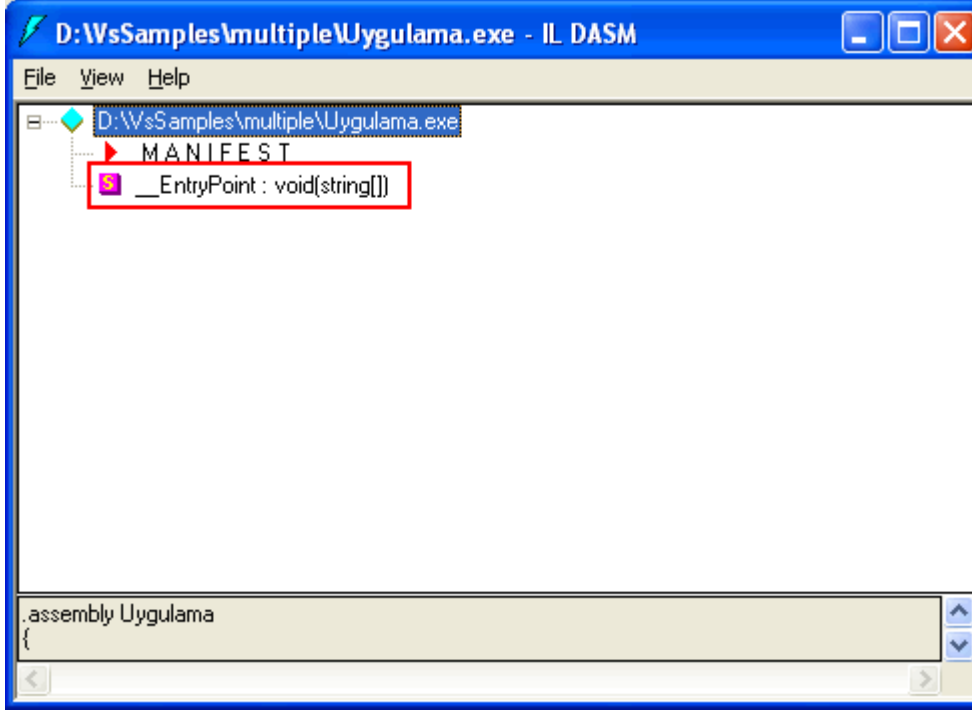
D:\UsSamples\multiple>_

```

Şekil 8. al (assembly linker) yardımıyla multiple-assembly çatısının oluşturulması.

Burada dikkat etmemiz gereken nokta main parametresi ile bildirdiğimiz yerdir. Burada, PE'imizin çalışmaya başlayacağı yerin, Temel isimli assembly'imız içindeki Main yordamı olduğunu belirtmiş oluyoruz. Elbette main parametresi, herhangi bir assembly içindeki Main yordamını belirtmek zorundadır. PE dosyamıza ayrıca, diğer kullanacağı assembly'lerden de bağlamış olduk. Şimdi Uygulama.exe assembly'imızın yapısını ildasm ile yakından inceleyelim.

Öncelikle ilk dikkati çeken nokta, entry point'in eklenmiş olmasıdır. Yazmış olduğumuz diğer assembly'lardan ne vb.net module assembly'imız ne j# kodlu kütüphane assembly'imızı nede PE'imizi oluşturan C# netmodule assembly'imızı (Main metodunu içermesine rağmen) herhangi bir entry point içermemekteydi. Ancak al (assembly linker) aracı ile tüm bu assembly'leri tek bir assembly altında birleştirirken, entry point'imizde Temel.netmodule assembly'ini içindeki Main yordamı olarak işaretlemiş olduk.



Şekil 9. PE olan assembly'ımızın içeriği.

Gelelim içeriğe. İlk dikkat çeken nokta diğer assembly'ların başvuru bilgilerinin eklenmiş olmasıdır. Başvuruların olduğu assembly'lar extern anahtar kelimesini içeren satırlardır. Extern anahtar kelimesi, bu assembly'ların PE tarafından erişilen ve kullanılan assembly'lar olduğunu gösterir. Bunun dışındaki tüm satırları ve bilgileri ile birlikte bu assembly aslında bir single-file assembly'dan farklıdır. Ancak kullandığı diğer assembly'lar düşünüldüğünde ortaya bir multiple-file assembly çıkmıştır.

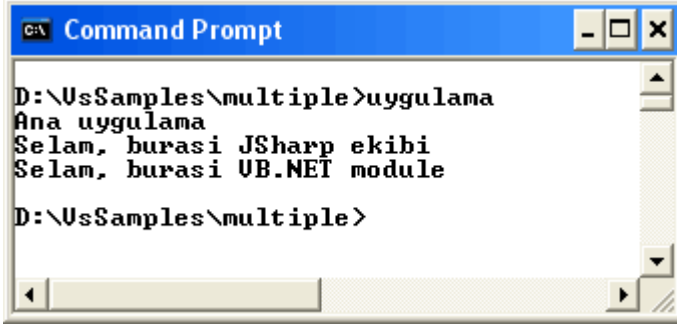
```
.module extern Temel.netmodule
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
  .hash = (4E FE C2 93 5B 46 10 72 20 30 9A 9C 31 21 D0 2F // N...[F.r 0..1!./
9B 84 AF 0E )
  .ver 1:0:3300:0
}
.assembly extern Microsoft.VisualBasic
{
  .publickeytoken = (B0 3F 5F 7F 11 D5 0A 3A ) // .?_.....:
  .ver 7:0:3300:0
}
.assembly extern jsharpselam
{
  .hash = (FA 10 B8 EE 98 5A F7 81 4F 0D 91 80 38 9D FB 78 // .....Z..O...8..x
```

```

04 14 A0 83 )
.ver 0:0:0:0
}
.assembly Uygulama
{
// --- The following custom attribute is added automatically, do not uncomment -----
// .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute::.ctor(bool,
// bool) = ( 01 00 00 01 00 00 )
.hash algorithm 0x00008004
.ver 0:0:0:0
}
.file vbselam.netmodule
.hash = (85 65 08 DF 98 6C 95 7F 61 47 8F 6C 02 DA 04 64 // .e...l..aG.l...d
B6 AC A2 F9 )
.file Temel.netmodule
.hash = (89 B4 FA 26 C6 7C 59 23 DB 92 6F 3A 29 5E 31 C7 // ...&.|Y#..o:)^1.
E7 37 35 AF ) // .75.
.class extern public vbdotnet.vbselam
{
.file vbselam.netmodule
.class 0x02000002
}
.class extern public Temel
{
.file Temel.netmodule
.class 0x02000002
}
.module Uygulama.exe
// MVID: {2F568429-2D83-4AA8-9558-46FB59574BBB}
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 512
.corflags 0x00000001
// Image base: 0x07000000

```

Uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Şekil 10. Uygulamanın çalışmasının sonucu.

Böylece geldik bir makalemizin daha sonuna. Umuyorum ki assembly'ları bu yönleri ile incelemek siz değerli okurlarımızın işine yarıyordur. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

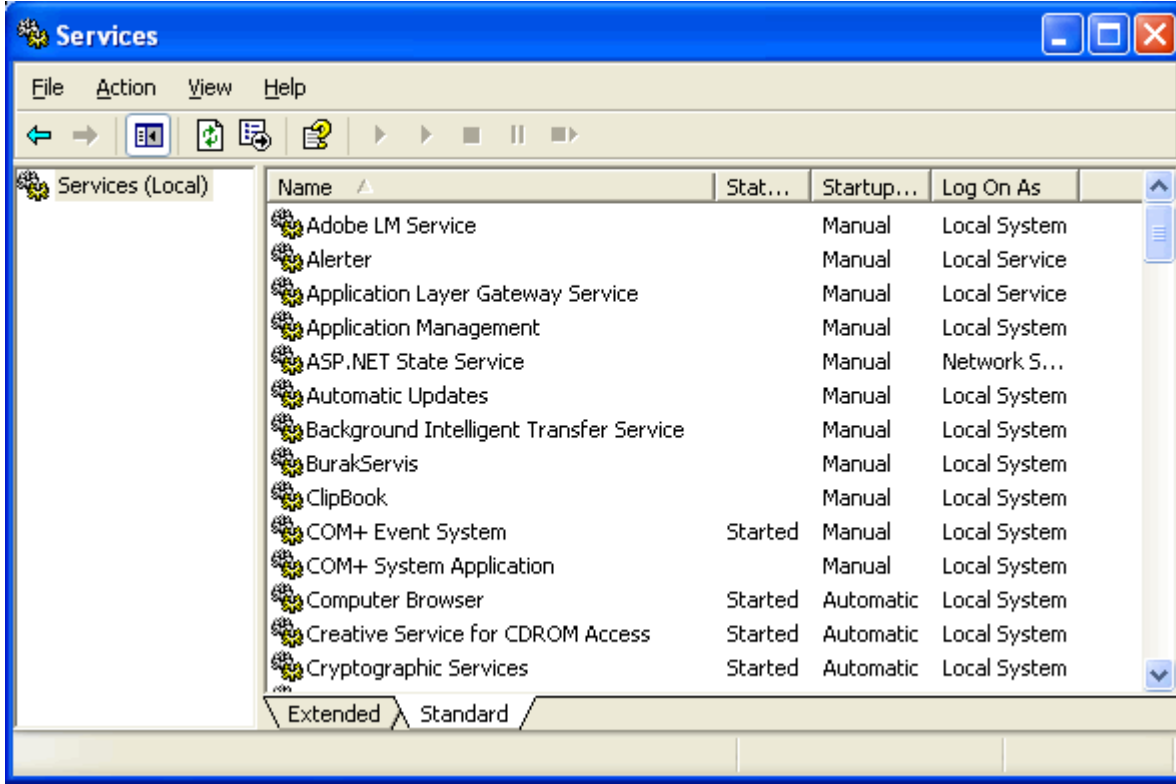
Windows Servislerine Giriş

Değerli Okurlarım, Merhabalar.

Bu makalemizde windows servislerine kısa bir giriş yapıcak ve en basit haliye bir windows servisinin, .net ortamında nasıl oluşturulacağını incelemeye çalışacağız. Öncelikle Windows Service nedir, ne amaçlarla kullanılır bunu irdelemeye çalışsak daha sonra windows servislerinin mimarisini kısaca inceleyeceğiz.

Windows servisleri, işletim sisteminde arka planda çalışan, kullanıcı ile etkileşimde bulunduğu herhangi bir arayüze sahip olmayan, kaynakların izlenmesi, system olaylarının log olarak tutulması, network erişimlerinin izlenmesi, veritabanları üzerindeki transaction'ların izlenmesi, sistem performansına atı bilgilerin toplanması, sistem hatalarının (system exceptions) , başarısız program denemelerin (failure) vb. gibi geri plan işlemlerinin takip edilmesinde kullanılan, sisteme kayıt edilmiş (register), çalıştırılabilir nesnelerdir.

Aslında, windows NT,2000,XP yada 2003 kullanıcısı iseniz, windows servisleri ile mutlaka ilgilenmişsinizdir. Sistemlerimizde çalışan pek çok servis vardır. Bu servislerin neler olduğuna, Administrative Tool bölümünde Services kısmından bakabiliriz. Örneğin aşağıda XP Professional sisteminde yüklü olan örnek servisler yer almaktadır.



Şekil 1. Win XP için Örnek Windows Servisleri

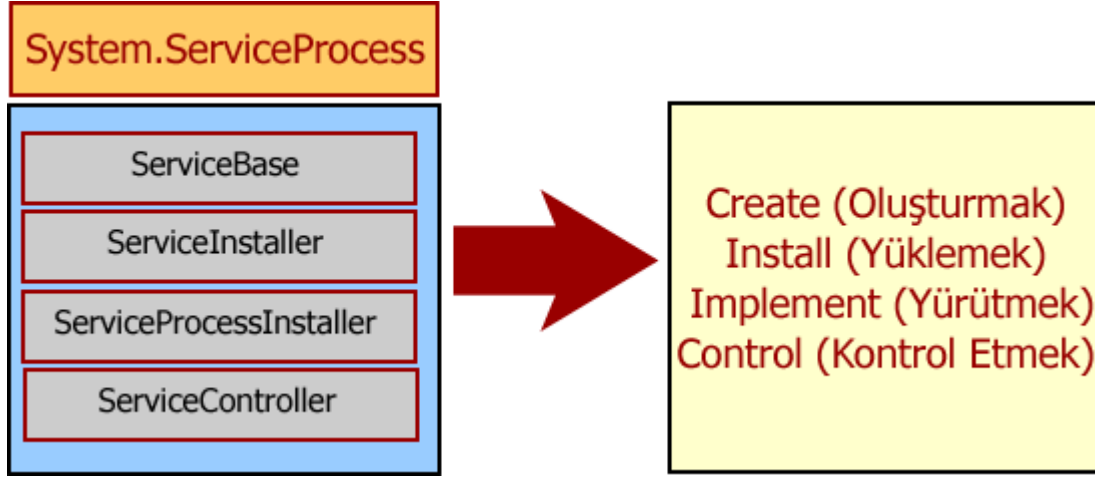
İşte biz, .net sınıflarını kullanarak, burada yer alacak windows servisleri yazma imkanına sahibiz. Windows Servislerinin mimari yapısı aşağıdaki şekilde görüldüğü gibidir.



Şekil 2. Windows Servis Mimarisi

Mimariden kısaca bahsetmek gerekirse; Service Application (Servis Uygulaması) , istenilen fonksiyonelliklere sahip bir veya daha fazla windows servisini içeren bir uygulamadır. Servis Kontrol Uygulaması (Service Controller Application) ise, servislerin davranışlarını kontrol eden bir uygulamadır. Son olarak, SCM, sistemde yüklü olan servislerin kontrol edilmesini sağlayan bir windows aracıdır. Dolayısıyla biz, bir windows servis uygulaması yazarken, bunun içerisine birden fazla servis koyabiliriz. Bu servis uygulaması ve başka servis uygulamalarının davranışlarını servis kontrol uygulamaları yardımı ile kontrol edebiliriz. Diğer yandan, yazmış olduğumuz tüm servis uygulamaları ile birlikte sistemdeki servisleri, SCM aracılığıyla yönetebiliriz.

.Net Framework, windows servislerini oluşturabilmemiz için gerekli sınıfları içeren System.ServiceProcess isim alanına (namespace) sahiptir. Bu isim alanındaki sınıfları kullanarak, bir servisi oluşturabilir, sisteme yükleyebilir, yürütebilir ve kontrol edebiliriz. Aşağıdaki şekil, basit olarak, ServiceProcess isim alanındaki sınıflar ile yapabileceklerimizi temsil etmektedir.

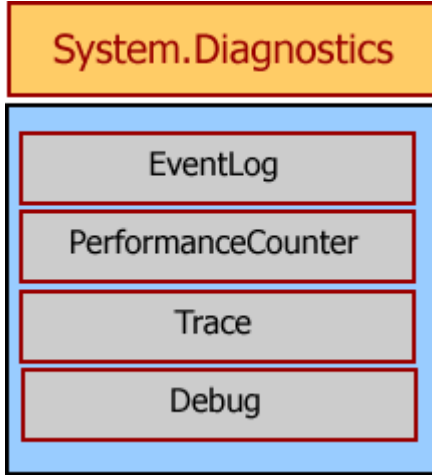


Şekil 3. System.ServiceProcess isim alanındaki sınıflar ile yapabileceklerimiz.

Buradaki sınıflar yardımı ile bir windows servisi oluşturmak istediğimizde izlememiz gereken bir yol vardır. Öncelikle, servisi oluşturmamız gerekir (Create) . Bunun için ServiceBase sınıfını kullanırız. ServiceBase sınıfında yer alan metodlar yardımıyla, bir windows servisini oluşturabiliriz. Oluşturulan bu servisin daha sonra, kullanılabilmesi için sisteme install edilmesi ve register olması gerekmektedir. Bu noktada devreye ServiceInstaller ve ServiceProcessInstaller sınıfları girer. Bir windows servis uygulamasını install etmeden önce, bu servis için bir iş parçacığı(process) oluşturulmalı ve yüklenmelidir. İşte bu noktada devreye ServiceProcessInstaller girer. ServiceInstaller ve ServiceProcessInstaller sınıfları aslında bir servisin sisteme yüklenebilmesi için gerekli metodları otomatik olarak sağlarlar. Ancak bu sınıfların uygulandığı bir windows servis uygulamasının tam olarak sisteme yüklenmesi ve Services kısmında görülebilmesi için, InstallUtil isimli .net aracı kullanılır ve oluşturulan windows servis uygulaması sisteme yüklenir.

Sisteme yüklenmiş olan servislerin kontrol edilebilmesi amacıyla, ServiceController sınıfındaki metodları kullanabiliriz. Yani, bir servisin Start, Stop, Pause, Continue gibi davranışlarını kontrol edebiliriz. Bu amaçla SCM aracını kullanabileceğimiz gibi, ServiceController sınıfındaki metodlarda kullanabilir ve böylece herhangi bir uygulamadan bir servisi başlatabilir, durdurabilir vb. işlemlerini gerçekleştirebiliriz.

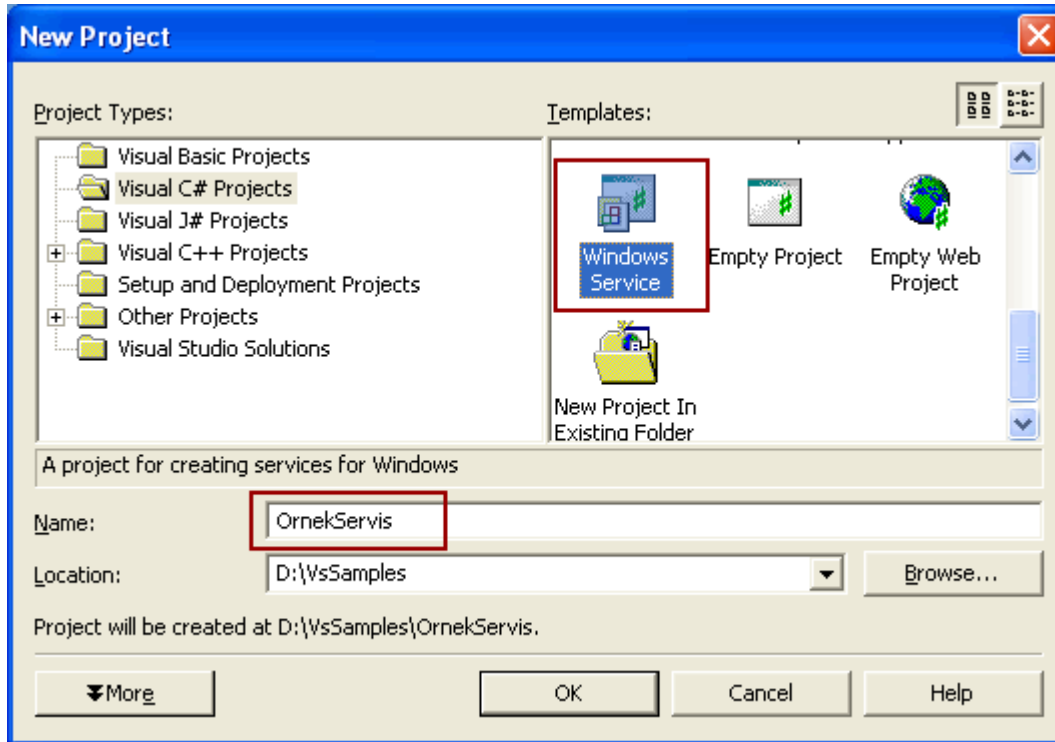
Bir windows servisinin oluşturulması, sisteme yüklenmesi , yürütülmesi ve kontrol edilmesi her ne kadar karışık görünsede, vs.net burada gereksinimin duyduğumuz işlemlerin çoğunu bizim için otomatik olarak yapmaktadır. Windows servislerinin oluşturulmasında kullanılan System.ServiceProcess isim alanının yanında, servislerin durumunun izlenebilmesi, çeşitli performans kriterlerinin servis içerisinden kontrol edilebilmesi, sisteme ait logların servis içerisinde kullanılabilmesi gibi işlemleri gerçekleştirebileceğimiz sınıfları içeren System.Diagnostics isim alanı da vardır.



Şekil 4. System.Diagnostics isim alanı sınıfları.

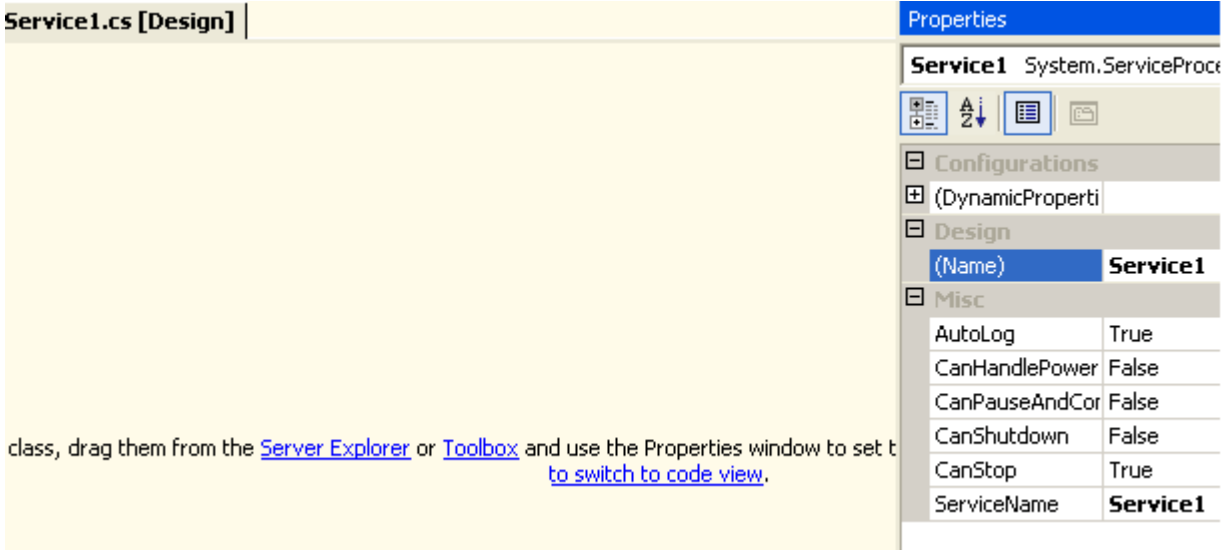
Bir windows servis uygulaması ile normal bir vs.net uygulaması arasında belirgin farklılıklar vardır. Herşeyden önce bir windows servis uygulamasının kullanılabilmesi için, sisteme install edilmesi ve register olması gereklidir. Diğer önemli bir farkta, bir windows servis uygulaması arka planda çalışırken, bu servisin çalışması ile ilgili oluşabilecek hatalardır. Normal bir windows uygulamasında hatalar sistem tarafından kullanıcıya bir mesaj kutusu ile gösterilebilir yada program içerisindeki hata kontrol mekanizmaları sayesinde görsel olarak izlenebilir. Oysa bir windows servis uygulaması çalışırken meydana gelen hatalar, event log olarak sistemde tutulurlar.

Windows servisleri ile ilgili bu kısa bilgilerin ardından, dilerseniz birlikte basit bir windows servis uygulaması geliştirelim. Bunun için ilk yapmamız gereken vs.net ortamında, bir windows service applicaiton projesi açmaktır.



Şekil 5. Windows Servis Uygulaması Açmak.

Bu işlemi gerçekleştirdiğimizde, vs.net herhangi bir kullanıcı arayüzü olmayan bir proje oluşturacaktır.



Şekil 6 . Windows Servis Uygulaması ilk oluşturulduğunda ekranın görünümü.

Servis uygulamasını geliştirmek için yazacağımız kodlara geçmek için, to switch to code windows linkine tıklamamız gerekiyor. Bu durumda vs.net tarafından otomatik olarak oluşturulmuş kodları görürüz. İlk dikkat çekici nokta, Service1 isimli sınıfın ServiceBase sınıfından türetilmiş olmasıdır.

```
public class Service1 : System.ServiceProcess.ServiceBase
```

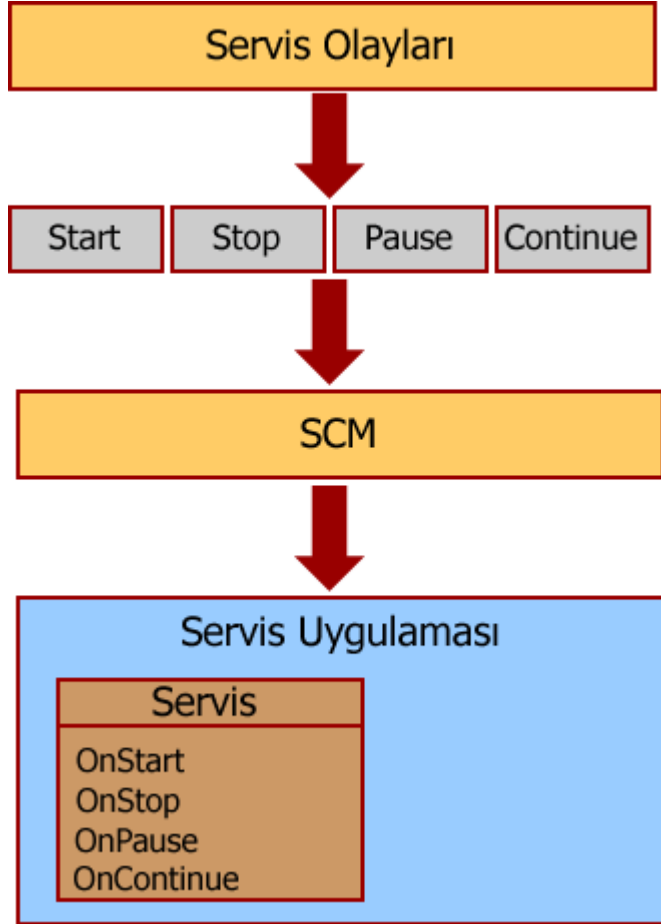
ServiceBase sınıfı OnStart, OnStop, OnPause, OnContinue gibi bir takım metodlar içeren bir sınıftır. Bir windows servisi oluşturulurken bu servis sınıfının, ServiceBase sınıfından türetilmesinin sebebi, bahsetmiş olduğumuz metodların, servis sınıfı içerisinde override edilmeleridir. Bir servis SCM tarafından yada windows içinden başlatıldığında, servisin başlatıldığında dair start mesajı gelir. İşte bu noktada servisin OnStart metodundaki kodlar devreye girer. Aynı durum benzer şekilde, OnStop olayı içinde geçerlidir. Tüm bu metodlar ve başka üyeler temel sınıf olan ServiceBase sınıfı içerisinde toplanmıştır. Bunun sonucu olarak, bir sınıfı servis olarak tanımlamışsak, ServiceBase sınıfından türetir böylece servis başlatıldığında, durdurulduğunda vb. olaylarda yapılmasını istediğimiz işlemleri, türeyen sınıfta bu davranışların tetiklediği, OnStart, OnStop gibi metodları override ederek gerçekleştirebiliriz.

Sonuç itibarıyla vs.net, bir servis uygulaması oluşturduğumuzda, buradaki varsayılan servisi, ServiceBase sınıfından türetir ve otomatik olarak OnStart ve OnStop metodlarını aşağıdaki şekilde ekler.

```
protected override void OnStart(string[] args)
{
}
}
```

```
protected override void OnStop()
{
}
}
```

Peki bu metodlar ne zaman ve ne şekilde çalışır. Bir windows servisinin, normal windows uygulamalarında olduğu gibi desteklediği olaylar vardır. Windows servisleri 4 olayı destekler. Bir windows servisinde meydana gelen olayların çalışma şekli aşağıdaki şekilde gibidir.



Şekil 7. Servis Olaylarının Ele Alınış Mekanizması.

Buradan görüldüğü gibi, sistemde, yazmış olduğumuz windows servisi ile ilgili olarak 4 olay meydana gelebilir. Bunlar Start, Stop, Pause ve Continue olaylarıdır. Bir servis SCM tarafından başlatıldığında bu servis için Start olayı meydana gelir. Daha sonra SCM, servisin içinde bulunduğu Servis Uygulamasına Start komutunu gönderir. Buna karşılık servis uygulaması içindeki ilgili servis bu Start komutunu alır ve karşılığında, OnStart metodunda yazan kodları çalıştırır.

Diğer yandan bir servis durdurulduğunda, Stop olayı meydana gelir. Ancak bu noktada SCM Start tekniğinden farklı olarak hareket eder. SCM önce oluşan olayın sonucunda ilgili servis uygulaması içindeki servisin, CanStop özelliğine bakar. CanStop özelliği true veya false değer alabilen bir özelliktir ve servisin durdurulup durdurulamıyacağını dolayısıyla, bir Stop olayı meydana geldiğinde, servise atı OnStop metodunun çalıştırılıp çalıştırılmıyacağını belirtir. SCM bu nedenle Stop olayı meydana geldiğinde ilk olarak bu

özellik kontrol eder. Eğer özellik değeri true ise, service Stop komutunu gönderir ve sonuç olarak OnStop metodundaki kodlar çalıştırılır.

Stop tekniğinde gerçekleşen özellik kontrol işlemi, Pause ve Continue olayları içinde geçerlidir. Bu kez SCM, servisin CanPauseAndContinue özelliğinin boolean değerine bakar. Eğer bu değer true ise, service Pause komutunu veya Continue komutunu gönderir ve bunlara bağlı olan OnPause ve OnContinue metodlarındaki kodların çalıştırılmasını sağlar. Pause olayı bir servis duraklatıldığında oluşur. Continue olayı ise, pause konumunda olan bir servis tekrar çalışmasına devam etmeye başladığında oluşur.

Bir diğer önemli nokta oluşturulan service1 isimli sınıfın main metodundaki kodlardır.

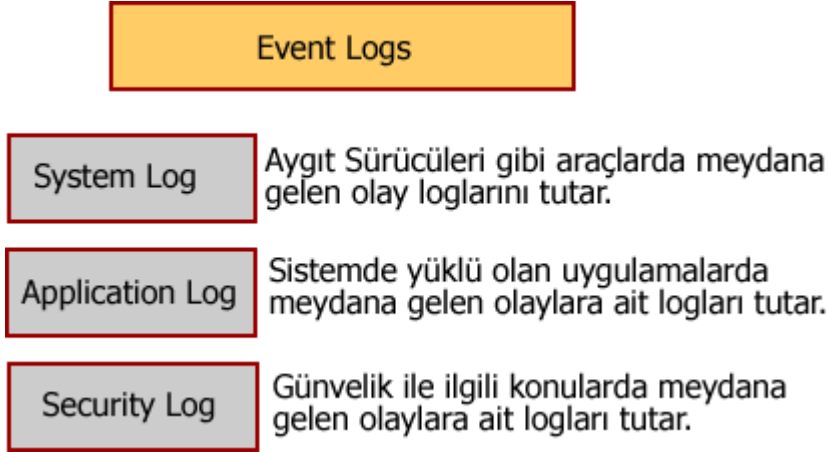
```
static void Main()
{
    System.ServiceProcess.ServiceBase[] ServicesToRun;
    ServicesToRun = new System.ServiceProcess.ServiceBase[] { new Service1() };
    System.ServiceProcess.ServiceBase.Run(ServicesToRun);
}
```

Burada görüldüğü gibi ServiceBase sınıfı tipinden bir nesne dizisi tanımlanmış ve varsayılan olarak bu nesne dizisinin ilk elemanı Service1 ismi ile oluşturulmuştur. Bu aslında bir servis uygulamasının birden fazla windows servisini içerebileceğinden göstermektedir. Burada ServiceBase sınıfı tipinden bir nesne dizisinin tanımlanmasının nedenide budur. Zaten oluşturduğumuz Service1 sınıfı, ServiceBase sınıfından türetildiği için ve sahip olduğu temel sınıf metodlarında bu türeyen sınıf içerisinde override edildiği için, polimorfizmin bir gereği olarak, servis nesnelerini bir ServiceBase tipinden dizide tutmak son derece mantıklı ve kullanışlıdır.

Son satıra gelince. ServiceBase sınıfının Run metodu, parametre olarak aldığı serviseleri, SCM tarafından başlatılmış iseler, belleğe yüklemekle görevli bir metoddur. Elbette servislerin belleğe Run metodu ile yüklenebilmesi için, öncelikle Start komutu ile başlatılmaları gerekmektedir. ServiceBase.Run metodu aslında normal bir windows uygulamasındaki Application.Run metodu ile aynı işlevselliği gösterir.

Şu ana kadar oluşturduğlarımız ile bir servisin omurgasını meydana çıkardık. Ancak halen servisimiz herhangi bir işlem yapmamakta. Oysaki bir servis ile, örneğin, sistemdeki aygıt sürücülerine ait olay log'larını, düşük bellek kullanımı sonucu yazılım veya donanım ekipmanlarında meydana gelebilecek hata (error), istisna(exception), warning(uyarı) gibi bilgilere ait log'ları izleyebiliriz.

Şimdi dilerseniz ilk yazdığımız servis ile Uygulama Log'larının (Application Log) nasıl tutulduğunu incelemeye çalışalım. Aslında bizim izleyebileceğimiz üç tip log vardır. Bunlar, System Log'ları, Application Log'ları ve Security Log'larıdır.



Şekil 8. Olay Log'larının Türleri.

Normal şartlar altında, vs.net ortamında bir windows servis ilk kez oluşturulduğunda, servise ait AutoLog özelliği değeri true olarak gelir. Bu durumda Start,Stop,Pause ve Continue olayları meydana geldiğinde, servis sisteme kurulduğunda veya sistemden kaldırıldığında, servis uygulamasına ait loglar, Application Log'a otomatik olarak yazılırlar. Biz bu uygulamamızda kendi özel loglarımızı tutmak istediğimizden bu özelliğin değerini false olarak belirliyoruz.

Kendi olay loglarımızı yazabilmemiz için, EventLog nesnelere ihtiyacımız vardır. EventLog nesneleri System.Diagnostics isim alanında yer alan EventLog sınıfı ile temsil edilirler. Bu nedenle uygulamızda öncelikle bir EventLog nesne örneği oluşturmalıyız.

```
private System.Diagnostics.EventLog OlayLog;
```

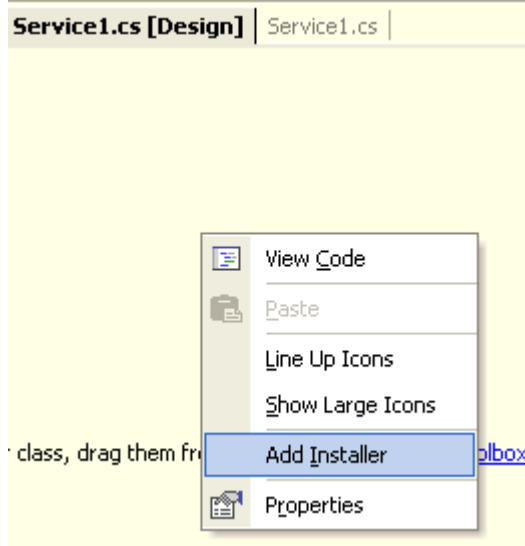
Bir EventLog nesnesi oluşturduktan sonra, bu nesne üzerinden CreateEventSource metodunu kullanarak servismiz için bir event log oluşturabiliriz. Buna ilişkin kodları OnStart metodu içine yazarsak, servis başlatıldığında oluşturduğumuz log bilgilerinde otomatik olarak yazılmasını sağlamış oluruz.

```
protected override void OnStart(string[] args)
```

```
{
    OlayLog=new EventLog(); /* EventLog nesnemizi olusturuyoruz.*/

    if(!System.Diagnostics.EventLog.SourceExists("Kaynak"))
    {
        System.Diagnostics.EventLog.CreateEventSource("Kaynak","Log Deneme"); /* İlk parametre ile, Log Deneme ismi altında
        tutulacak Log bilgilerinin kaynak ismi belirleniyor. Daha sonra bu kaynak ismi OlayLog isimli nesnemizin Source özelligine
        ataniyor.*/
    }
    OlayLog.Source="Kaynak";
    OlayLog.WriteEntry("Servisimiz baslatildi...",EventLogEntryType.Information); /* Log olarak ilk parametrede belirtilen mesaj
    yazilir. Log'un tipi ise ikinci parametrede görüldüğü gibi Information'dir.*/
}
```

Bu işlemlerin ardından servisimiz için basit bir görevide eklemiş olduk. Şimdi sırada bu servisin sisteme yüklenmesi var. Bunun için, daha önceden bahsettiğimiz gibi, ServiceProcess.ServiceInstaller ve ServiceProcess.ServiceProcessInstaller sınıflarını kullanmamız gerekiyor. Servis uygulamamız için bir tane ServiceProcessInstaller sınıfı nesne örneğine ve servis uygulaması içindeki her bir servis içinde ayrı ayrı olmak üzere birer ServiceInstaller nesne örneğine ihtiyacımız var. Bu nesne örneklerini yazmış olduğumuz servis uygulamasına kolayca ekleyebiliriz. Bunun için, servisimizin tasarım penceresinde sağ tuşa basıyor ve Add Installer seçeneğini tıklıyoruz.



Şekil 9. Add Installer.

Bunun sonucu olarak vs.net, uygulamamıza gerekli installer sınıflarını yükler.



Şekil 10. Installer nesne örneklerinin yüklenmesi.

Bunun sonucu olarak aşağıda görülen sınıf uygulamamıza otomatik olarak eklenir. Bu aşamada buradaki kodlar ile fazla ilgilenmeyeceğiz.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Configuration.Install;

namespace OrnekServis
{
    [RunInstaller(true)]
    public class ProjectInstaller : System.Configuration.Install.Installer
```

```

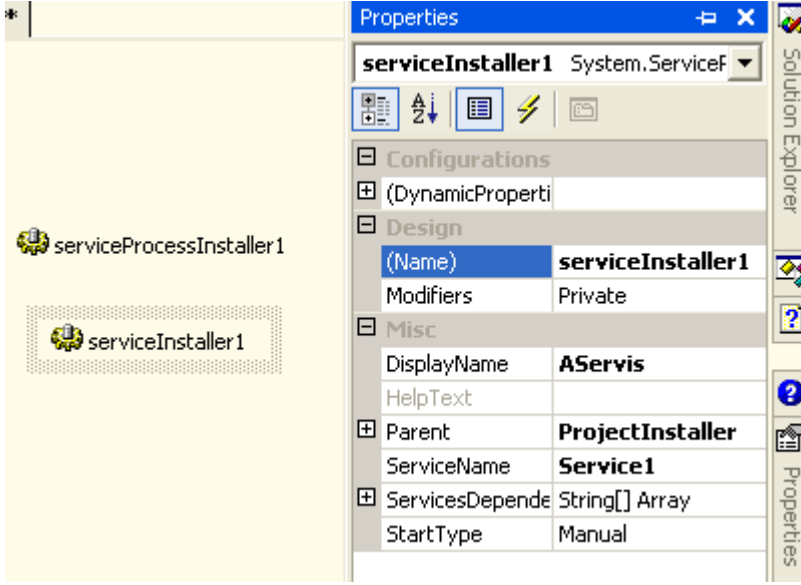
{
    private System.ServiceProcess.ServiceProcessInstaller serviceProcessInstaller1;
    private System.ServiceProcess.ServiceInstaller serviceInstaller1;
    private System.ComponentModel.Container components = null;

    public ProjectInstaller()
    {
        InitializeComponent();
    }
    protected override void Dispose( bool disposing )
    {
        if( disposing )
        {
            if(components != null)
            {
                components.Dispose();
            }
        }
        base.Dispose( disposing );
    }
    private void InitializeComponent()
    {
        this.serviceProcessInstaller1 = new System.ServiceProcess.ServiceProcessInstaller();
        this.serviceInstaller1 = new System.ServiceProcess.ServiceInstaller();
        this.serviceProcessInstaller1.Password = null;
        this.serviceProcessInstaller1.Username = null;
        this.serviceInstaller1.ServiceName = "Service1";
        this.Installers.AddRange(new System.Configuration.Install.Installer[] {this.serviceProcessInstaller1,this.serviceInstaller1});

    }
}

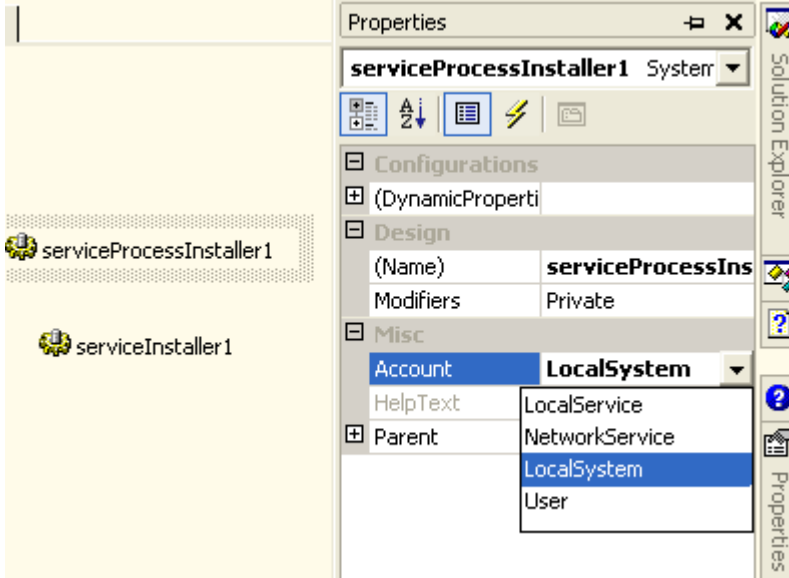
```

Ancak servisimizin, Sistemdeki service listesinde nasıl görüneceğini belirlemek için, ServiceInstaller nesne örneğinin, Display Name özelliğini değiştirebiliriz.



Şekil 11. Servisimizin görünen adını değiştiriyoruz.

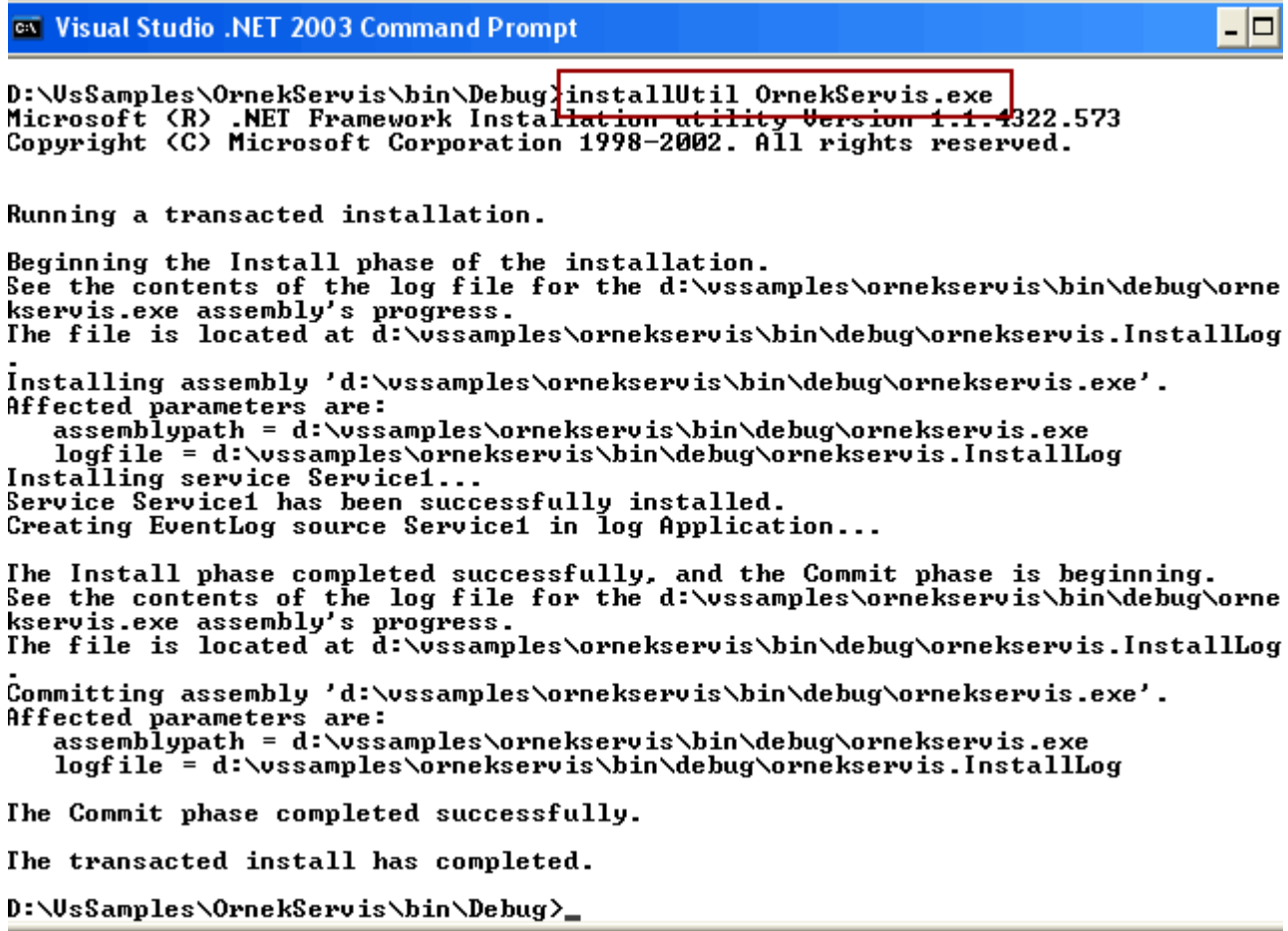
Diğer taraftan yapmamız gereken bir işlem daha var. ServiceProcessInstaller nesne örneğinin Account özelliğinin değerini belirlemeliyiz. Bu özellik aşağıdaki şekilde görülen değerlerden birisini alır.



Şekil 12. Account Özelliğinin Alabileceği Değerler.

Biz bu uygulama için LocalSystem değerini veriyoruz. Bu, Servis Uygulamamızın sisteme yüklenirken, sistemdeki kullanıcıların özel haklara sahip olmasını gerektirmez. Dolayısıyla sisteme giren her kullanıcının servisi yükleme hakkı vardır. Eğer User seçeneğini kullanırsak, servisin sisteme yüklenebilmesi için geçerli bir kullanıcı adı ve parolanın girilmesi gerekmektedir.

Artık yazmış olduğumuz servis uygulaması için installer'larda oluşturduğumuza göre uygulamamız derleyip, sisteme InstallUtil aracı ile yükleyebilir ve register edebiliriz. Bunun için, servis uygulamamızın exe dosyası ile, InstallUtil aracını aşağıdaki gibi kullanırız.



```

C:\ Visual Studio .NET 2003 Command Prompt

D:\UsSamples\OrnekServis\bin\Debug>installUtil OrnekServis.exe
Microsoft (R) .NET Framework Installation Utility Version 1.1.4322.573
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Running a transacted installation.

Beginning the Install phase of the installation.
See the contents of the log file for the d:\vssamples\ornekservis\bin\debug\ornekservis.exe assembly's progress.
The file is located at d:\vssamples\ornekservis\bin\debug\ornekservis.InstallLog

Installing assembly 'd:\vssamples\ornekservis\bin\debug\ornekservis.exe'.
Affected parameters are:
  assemblypath = d:\vssamples\ornekservis\bin\debug\ornekservis.exe
  logfile = d:\vssamples\ornekservis\bin\debug\ornekservis.InstallLog
Installing service Service1...
Service Service1 has been successfully installed.
Creating EventLog source Service1 in log Application...

The Install phase completed successfully, and the Commit phase is beginning.
See the contents of the log file for the d:\vssamples\ornekservis\bin\debug\ornekservis.exe assembly's progress.
The file is located at d:\vssamples\ornekservis\bin\debug\ornekservis.InstallLog

Committing assembly 'd:\vssamples\ornekservis\bin\debug\ornekservis.exe'.
Affected parameters are:
  assemblypath = d:\vssamples\ornekservis\bin\debug\ornekservis.exe
  logfile = d:\vssamples\ornekservis\bin\debug\ornekservis.InstallLog

The Commit phase completed successfully.

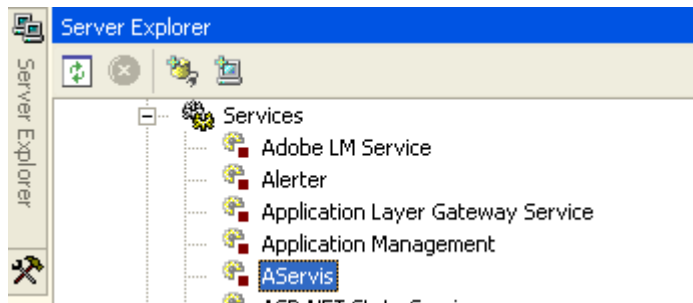
The transacted install has completed.

D:\UsSamples\OrnekServis\bin\Debug>_

```

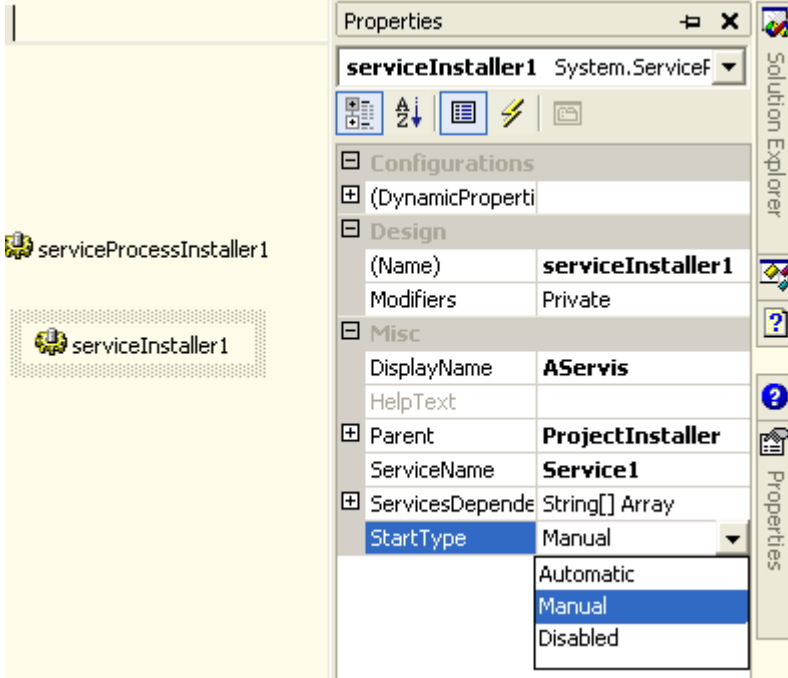
Şekil 13. InstallUtil aracı yardımıyla bir servisin sisteme yüklenmesi.

İşte InstallUtil aracı servisimizi sisteme yüklerken, servis uygulamamıza eklediğimiz ServiceProcessInstaller ve ServiceInstaller sınıflarını kullanır. Bu işlemin ardından vs.net ortamında, Server Explorer'dan servislere baktığımızda, AServis isimli servisimizin yüklendiğini ancak henüz çalıştırılmadığını görürüz.



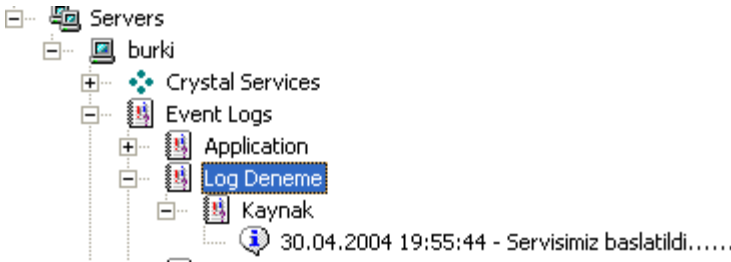
Şekil 14. Servis sisteme yüklendi.

Eğer servisin otomatik olarak başlatılmasını istiyorsak, ServiceInstaller nesnesinin StartType özelliğini Automatic olarak ayarlamamız yeterli olacaktır.



Şekil 15. Servisin Başlatılma Şeklinin Belirlenmesi.

İstersek servisimizi, yine Server Explorer'dan servis adına sağ tıklayıp açılan menüden start komutuna basarak çalıştırabiliriz. Servisimizi çalıştırdıktan sonra, yine Server Explorer pencersinden Event Logs sekmesine bakarsak, bahsetmiş olduğumuz olay logları haricinde kendi yapmış olduğumuz olay logunda eklendiğini görürüz.



Şekil 16. Servisin çalışmasının sonucu.

Burada görüldüğü gibi Log Deneme ismi ile belirttiğimiz Event Log oluşturulmuş, Source olarak belirttiğimiz Kaynak nesnesi oluşturulmuş ve OnStart metoduna yazdığımız kodlar yardımıyla, mesaj bilginiz information(bilgi) olarak yazılmıştır. Buraya kadar anlattıklarımız ile bir windows servisinin nasıl yazıldığını ve sisteme yüklendiğini en temek hatları ile görmüş olduk. İlerleyen makalelerimizde, windows servisleri ile ilgili daha farklı uygulamalar geliştirmeye çalışacağız. Hepinize mutlu günler dilerim.

Windows Servislerinin Kontrolü -1

Değerli Okurlarım, Merhabalar.

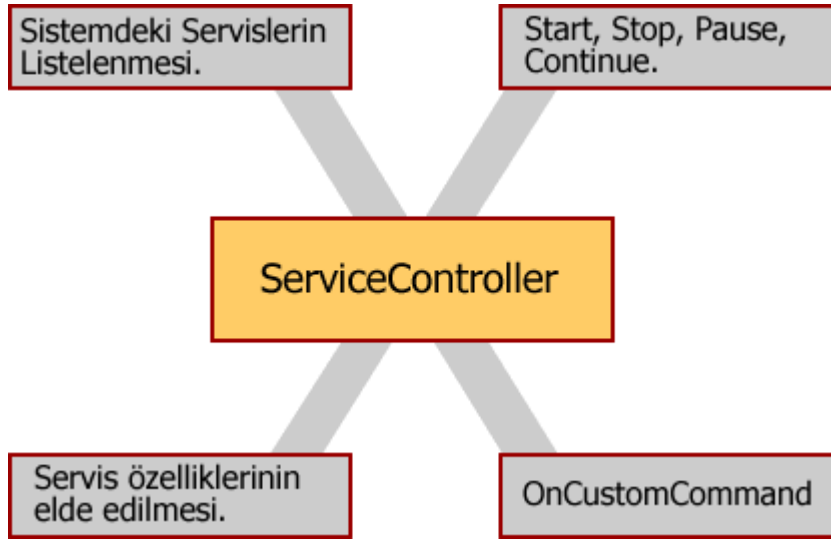
Bu makalemizde, windows servislerinin, bir windows uygulamasından nasıl kontrol edilebileceğini incelemeye çalışacağız. Bir önceki makalemizde, windows servislerinin nasıl oluşturulduğunu ve sisteme nasıl yüklendiklerini incelemiştik. Oluşturduğumuz windows servislerini(sistemdeki windows servislerini), SCM yardımıyla yönetebilmekteyiz. Ancak dilersek, bu yönetimi programlarımız

içindende gerçekleştirebiliriz. Bunu sağlayabilmek için, System.ServiceProcess isim alanında yer alan ServiceController sınıfını ve üyelerini kullanmaktayız.

ServiceController sınıfı ile windows servislerine bağlanabilir ve onları kontrol edebiliriz. Örneğin servisleri başlatabilir, durdurabilir veya sistemdeki servisleri elde edebiliriz. Bu ve benzeri olanaklar dışında SCM ile yapamıyacağımız bir olayı da gerçekleştirebiliriz. Bu olay windows servislerinin OnCustomCommand metodu üzerinde işlemektedir. Bir windows servisinin OnCustomCommand metodu sayesinde servisimize standart işlevselliklerinin haricinde yeni işlevsellikler kazandırabiliriz. Prototipi aşağıdaki gibi olan OnCustomCommand metodu integer tipinden bir parametre almaktadır.

protected virtual void OnCustomCommand(int command);

OnCustomCommand metodunu herhangi bir windows uygulamasından çağırabilmek için, ServiceController sınıfının ExecuteMethod metodu kullanılır. Integer tipindeki parametre 128 ile 256 arasında sayısal değerler alır. Metoda gönderilen parametre değerine göre, OnCustomCommand metodunun farklı işlevsellikleri yerine getirmesini sağlayabiliriz. ServiceController sınıfı ile yapabileceklerimiz aşağıdaki şekilde kısaca özetlenmiştir.



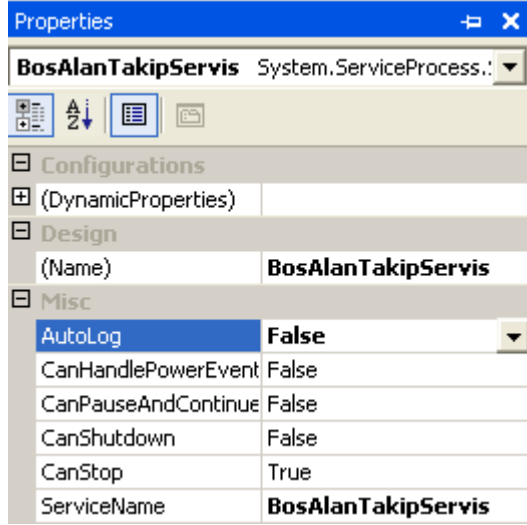
Şekil 1. ServiceController Sınıfı İle Yapabileceklerimiz.

Servislerimizi programatik olarak nasıl kontrol edebileceğimize geçmeden önce, konumuz ile ilgili basit bir windows servisi yazarak işe başlayalım. Bu servisimizde, kendi oluşturacağımız bir Event Log içerisinde, sistemdeki sürücülerinin boş alan kapasitelerine ait bilgileri tutacağız. Bu amaçla, servisimize bir Performance Counter ekleyeceğiz. Ayrıca servisimize bir timer kontrolü koyacak ve bu kontrol ile belirli periyotlarda servisin, boş alan bilgilerini kontrol etmesini ve belli bir serviyenin altına inilirse Event Log'umuza bunu bir uyarı simgesi ile birlikte yazmasını sağlayacağız.

Bununla birlikte, OnCustomCommand metodunu uygulamamızdan çalıştıracak ve gönderdiğimiz parametre değerine göre servisin değişik işlevleri yerine getirmesini sağlayacağız. Örneğin kullanıcının boş alan kontrol süresini ve alt sınır değerlerini programatik olarak belirleyebilmesini ve servisteki ilgili değerleri buna göre değiştirebilmesini sağlayacağız. Elbette bu değişiklikler servisin çalışma süresi boyunca geçerli olacaktır.

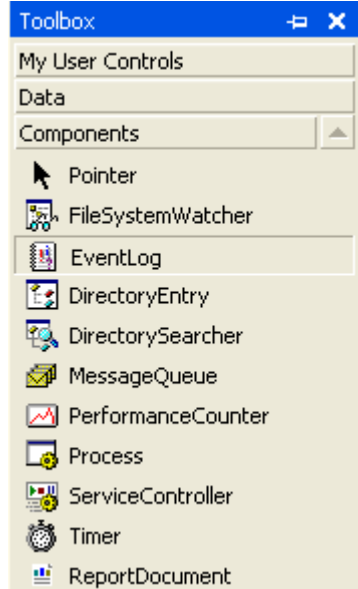
Şimdi dilerseniz servisimizi oluşturalım. Öncelikle vs.net ortamında, yeni bir windows service projesi açıyoruz. Projemizin adı, BosAlanTakip olsun. Ardından servisimizin özelliklerinden adını ve servise ait kodlarda yer alan Service1'i, BosAlanTakipServis

olarak değiştirelim. Bununla birlikte Soluion Explorer'da Service1.cs kod dosyamızın adınıda BosAlanTakipServis.cs olarak değiştirelim. Sonraki adımımız ise, servisin özelliklerinden AutoLog özelliğine false değerini atamak. Nitekim biz bu servisimizde kendi yazacağımız Event Log'u kullanmak istiyoruz.



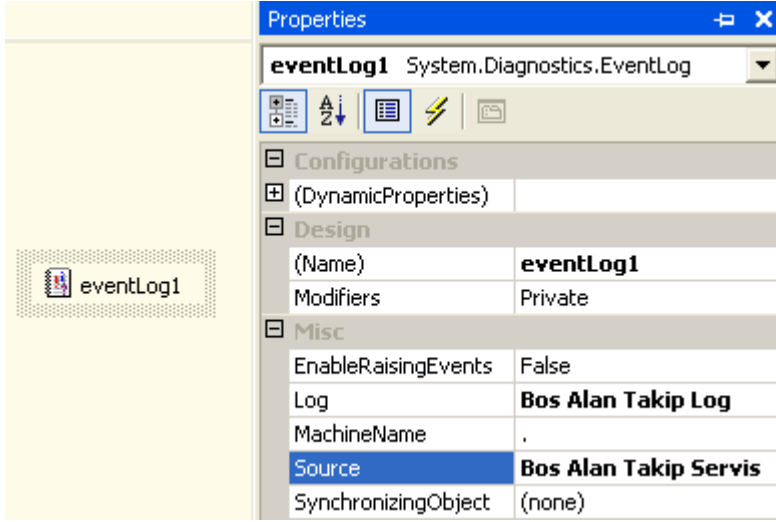
Şekil 2. Servisimizin özelliklerinin belirlenmesi.

Servisimizin kodlarını yazmadan önce, Custom Event Log'umuzu ekleyeceğiz. Bunun için, Components sekmesinden EventLog nesnesini servisimizin tasarım ekranına sürüklüyoruz.



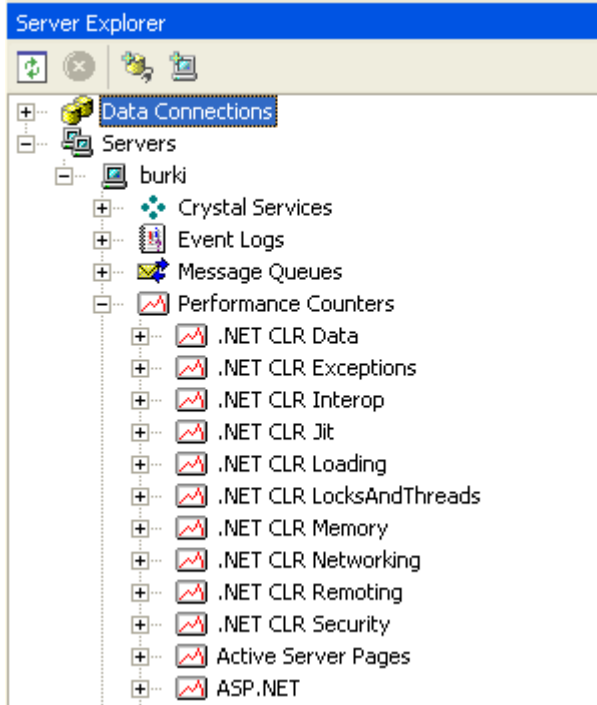
Şekil 3. EventLog nesnesi.

Şimdi EventLog nesnemizin özelliklerini belirleyelim.



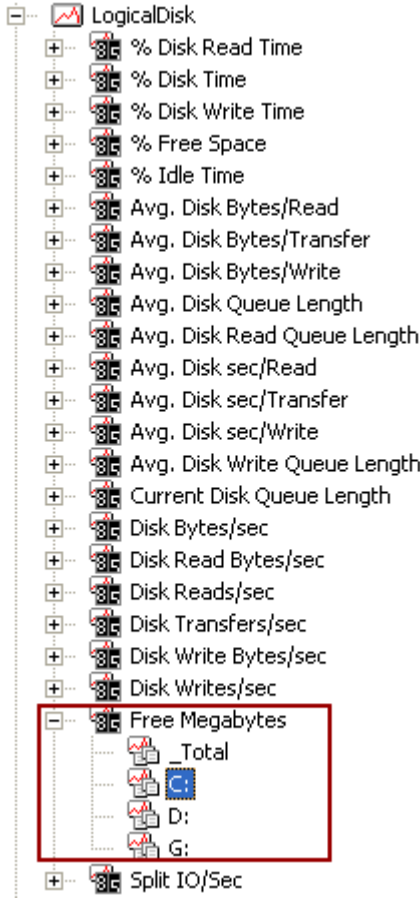
Şekil 4. EventLog nesnemin özellikleri.

Artık log bilgilerini, servisimize eklediğimiz eventLog1 nesnesini kullanarak oluşturacağız. Sırada servisimize bir Performance Counter nesnesi eklemek var. Servislerimizde kullanabileceğimiz Performance Counter öğelerini, Solution Explorer penceresinde yer alan Performance Counter sekmesinden izleyebiliriz. Sistemde yüklü olan pek çok Performance Counter vardır.



Şekil 5. Sistemdeki Performance Counter'lerden bazıları.

Biz bu örneğimizde, LogicalDisk sekmesinde yer alan Free MegaBytes öğesini kullanacağız. Bu sayaç yardımıyla, servisimizden, sistemdeki bir hardisk'in boş alan bilgilerini elde edebileceğiz. Bu amaçla buradaki C: sürücüsünü servisimiz üzerine sürüklüyoruz.



Şekil 6. Kullanacağımız Performance Counter ögesi.

Diğer ihtiyacımız olan nesne bir Timer. Bunun için yine Components sekmesinden Timer nesnesini , servisimizin tasarım ekranına sürüklememiz yeterli. Daha sonra Timer nesnesinin interval özelliğinin değerini 600000 olarak değiştirelim. Bu yaklaşık olarak 10 dakikada (1000 milisaniye * 10 dakika * 60 saniye) bir, Timer nesnesinin Elapsed olayının çalıştırılacağını belirtmektedir. Biz bu olay içerisinde C sürücüsündeki boş alan miktarını kontrol etmeyi planlıyoruz. Elbette süreyi başlangıçta 10 dakika olarak belirlememize rağmen geliştireceğimiz windows uygulamasında bu sürenin kullanıcı tarafından değiştirilebilmesini sağlayacağız. Bu ayarlamaların ardından servisimiz için gerekli kodları yazmaya geçebiliriz.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.ServiceProcess;

namespace BosAlanTakip
{
    public class BosAlanTakipServis : System.ServiceProcess.ServiceBase
```

```

{
    ...
    private long BosMiktar;
    private long Sinir;

    /* Servisimiz çalıştırıldığında(start),durdurulduğunda(stop), duraksatıldığında(pause) ve yeniden çalıştırıldığında(continue) ,
    zaman bilgisini ve performanceCounter1 nesnesinin RawValue özelliğini kullanarak C sürücüsündeki boş alan miktarını,
    oluşturduğumuz Event Log'a WriteEntry metodu ile yazıyoruz. Servisin durum bilgisini ise metoda gönderdiğimiz string türünden
    parametre ile elde ediyoruz.*/
    private void Bilgilendir(string durum)
    {
        eventLog1.WriteEntry(durum+" "+DateTime.Now.ToShortDateString()+ "
C:"+performanceCounter1.RawValue.ToString()+" mb");
    }
    protected override void OnStart(string[] args)
    {
        Bilgilendir("START");
        BosMiktar=performanceCounter1.RawValue; /* Servis çalıştırıldığında, C sürücüsündeki boş alan miktarını, BosMiktar isimli
        long türünden değişkenimize aktarıyoruz. performanceCounter'ımızın RawValue özelliği burada seçtiğimiz sayaç kriteri gereği
        sonucu megabyte cinsinden döndürmektedir.*/
        Sinir=3300; // Yaklaşık olarak 3.3 GB.
        timer1.Enabled=true; // Timer nesnemizi çalıştırıyoruz.
    }
    protected override void OnStop()
    {
        Bilgilendir("STOP");
    }
    protected override void OnPause()
    {
        Bilgilendir("PAUSE");
    }
    protected override void OnContinue()
    {
        Bilgilendir("CONTINUE");
    }
    /* Özel komutumuzu yazıyoruz. */
    protected override void OnCustomCommand(int command)
    {
        /* if koşullarında OnCustomCommand'a gönderilecek parametre değerine göre, boş alan uyarısı için gerekli alt sınır değeri

```

belirleniyor. Ayrıca, timer nesnemizin interval değeri gelen parametre değerine göre belirleniyor ve böylece timer nesnesinin Elapsed olayının çalışma aralığı belirlenmiş oluyor.*/

```

        if(command==200)
        {
            Sinir=3000; // Yaklaşık olarak 3gb.
        }
        else if(command==201)
        {
            Sinir=2000; // Yaklaşık olarak 2gb.
        }
        else if(command==202)
        {
            Sinir=4000; // Yaklaşık olarak 4gb.
        }
        else if(command==203)
        {
            timer1.Enabled=false;
            timer1.Interval=1800000; // 30 dakikada bir.
            timer1.Enabled=true;
        }
        else if(command==204)
        {
            timer1.Enabled=false;
            timer1.Interval=3600000; // Saatte bir.
            timer1.Enabled=true;
        }
        else if(command==205)
        {
            timer1.Enabled=false;
            timer1.Interval=3000; // 3 Saniyede bir.
            timer1.Enabled=true;
        }
    }

    private void timer1_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
    {
        if(BosMiktar<=Sinir)
        {
            /* Eğer C sürücüsündeki boş alan miktarı Sinir değişkeninin sahip olduğu değerin altına düşerse, bu bilgi Event Log'umuza bir warning singesi ile birlikte eklenecek. */

```



```

        string bilgi=DateTime.Now.ToShortDateString()+ " C: Surucusu Bos Alan :
"+performanceCounter1.RawValue.ToString();
        eventLog1.WriteEntry(bilgi,EventLogEntryType.Warning);
    }
    else
    {
        Bilgilendir("");
    }
}
}
}
}

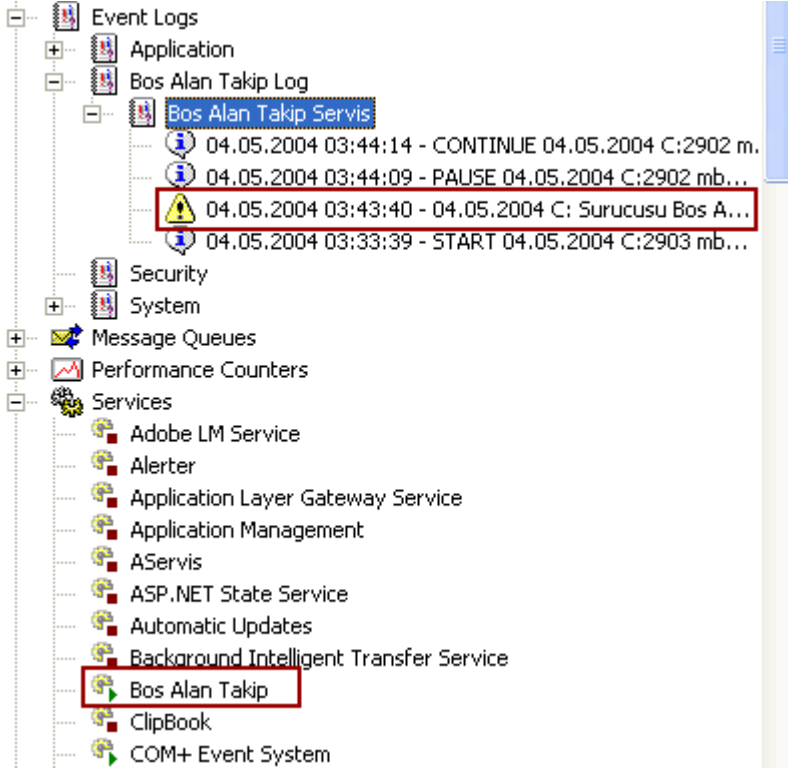
```

Servisimizin kodlarınıda böylece hazırlamış olduk. Şimdi servisimiz için gerekli olan installer'larımızı servisimizin tasarım ekranında sağ tuşla açtığımız menüden, Add Installer öğesini seçerek ekleyelim. Önce ServiceInstaller1'in özelliklerinden Display Name özelliğinin değerini, Bos Alan Takip olarak değiştirelim. Böylece servisimizin, services sekmesinde görünen ismini belirlemiş oluruz. Ardından ServiceProcessInstaller1 nesnemizin, Account özelliğinin değerini, LocalSystem olarak değiştirelim. Böylece sistemi açan herkes bu servisi kullanabilecek.

Diğer yandan oluşturduğumuz Custom Event Log içinde bir installer oluşturmamız gerekiyor ki installUtil tool'u ile servisimizi sisteme kurduğumuzda, sistedeki Event Log'lar içerisine, oluşturduğumuz Custom Event Log'da kurulabilsin. Bu amaçla, eventLog1 nesnemiz üzerinde sağ tuşa basıp çıkan menüden Add Installer'ı seçiyoruz. Bu sayede, Event Log'umuz için, bir adet eventLogInstaller'ın oluşturulduğunu görürüz. Bu işlemlerin ardından windows servis uygulamamızı derleyelim ve servisimizi sisteme yükleyebilmek için installUtil, vs.net tool'unu aşağıdaki gibi kullanalım.

installUtil BosAlanTakip.exe

Bu işlemlerin ardından servisimiz başarı ile yüklendiyse, server explorer'daki services sekmesinde görünecektir. Servisimizi bu aşamada denemek amacı ile, başlatıp durdurmayı deneyelim. Ancak bu noktada servisimizin pause ve continue olaylarının geçersiz olduğunu görürüz. Bunun sebebi, servisimizin OnPauseContinue özelliğinin değerinin false olmasıdır. Şimdi bu değeri true yapalım. Servis uygulamamızı tekrar derleyelim. Ardından servisimizi sisteme yeniden yükleyelim. Şimdi server explorer'dan servisimizi çalıştırıp deneyelim.



Şekil 7. Servisin Çalışan Hali.

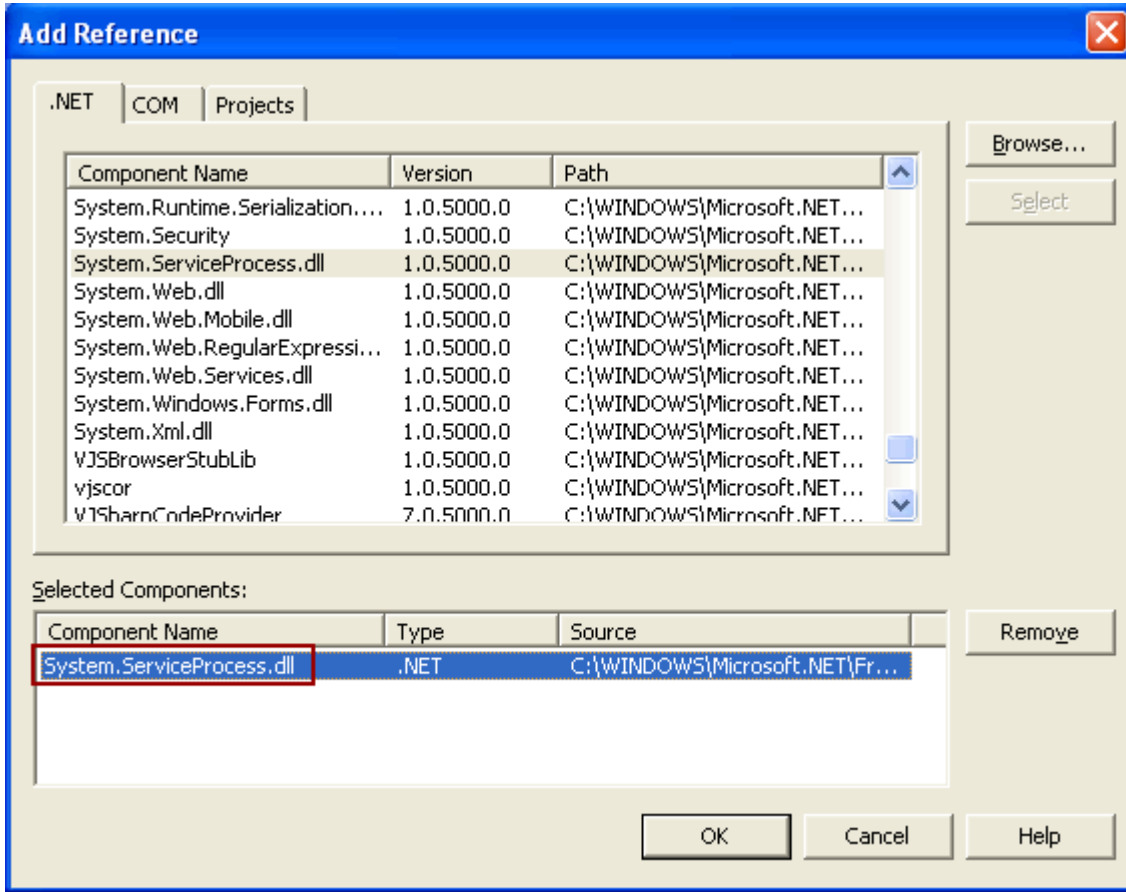
Ben servisi test etmek için, C sürücüsündeki boş alanı biraz daha düşürdüm. Görüldüğü gibi servis başarılı bir şekilde çalışıyor. Şu an için her 10 dakikada bir timer nesnesinin Elapsed olayı devreye giriyor ve C sürücüsündeki boş alan miktarının 3.3 gb'ın altına düşüp düşmediği kontrol ediliyor. Servisimiz bunu tespit ettiği anlarda Event Log'umuza bir uyarı işareti ekleyecektir.

Şimdi yazmış olduğumuz bu servisi başka bir windows uygulaması içerisinde nasıl yönetebileceğimizi incelemeye başlayalım. Burada bizim için anahtar nokta, windows uygulamamızda, System.ServiceProcess isim alanını kullanmak. Oluşturmuş olduğumuz servise ait OnCustomCommand metodunu çalıştırmak için, bu isim alanında yer alan ServiceController sınıfının ExecuteCommand metodunu kullanacağız. Öncelikle aşağıdaki gibi bir form tasarlayarak işe başlayalım.



Şekil 8. Form Tasarımımız.

Şimdi ServiceProcess isim alanını kullanabilmek için Add Reference kısmından uygulamamıza eklememiz gerekiyor.



Şekil 9. System.ServiceProcess isim alanının uygulamaya eklenmesi.

Öncelikle servisimizi temsil edecek ServiceController sınıfı türünden bir nesne oluşturacağız. ServiceController sınıfından nesnemizi oluştururken aşağıda prototipi verilen yapıcı metodu kullanıyoruz. Bu yapıcı, servisin tam adını parametre olarak string türünden almaktadır.

```
public ServiceController(string serviceName);
```

ServiceController sınıfına ait nesne örneğini kullanarak servisimize ilişkin pek çok bilgiyi elde edebiliriz. Örneğin, servisin görünen adını DisplayName özelliği ile , servisin çalıştığı makine adını MachineName özelliği ile elde edebiliriz. ServiceController sınıfının üyeleri yardımıyla sistemimizde kurulu olan servislere ait pek çok bilgiyi temin edebiliriz. Bu konuyu bir sonraki makalemizde incelemeye çalışacağız. Şimdi derseniz servisimizi kontrol edeceğimiz kısa windows uygulamamızın kodlarını yazalım.

```
ServiceController sc; /* Servisimizi kontrol edecek olan ServiceController sınıf nesnemiz tanımlanıyor.*/
```

```
private void Form1_Load(object sender, System.EventArgs e)
{
```

```
sc=new ServiceController("BosAlanTakipServis"); /* ServiceController nesnemiz, kullanmak istediğimiz servisin adını parametre olarak almak suretiyle oluşturuluyor.*/
```

```
lblServisAdi.Text=sc.DisplayName.ToString()+"/"+sc.MachineName.ToString(); /* Servisimizin , sistemdeki services kısmında görünen ismi ve üzerinde çalıştığı makine adı elde ediliyor ve label kontrolümüze ekleniyor.*/
```

```
}
```

```
private void btnPeriyod_Click(object sender, System.EventArgs e)
```

```
{
```

```
    if(lblPeriyod.SelectedIndex==0)
```

```
    {
```

```
        sc.ExecuteCommand(203); /* Servisimizdeki OnCustomCommand metodu çalıştırılıyor ve bu metoda parametre değeri olarak 203 gönderiliyor. Artık servisimiz, 203 parametre değerine göre bir takım işlevler gerçekleştirecek. 203 değeri karşılığında servisimizdeki OnCustomCommand metodu, log tutma süresini yarım saat olarak belirleyecektir.*/
```

```
    }
```

```
    else if(lblPeriyod.SelectedIndex==1)
```

```
    {
```

```
        sc.ExecuteCommand(204);
```

```
    }
```

```
    else if(lblPeriyod.SelectedIndex==2)
```

```
    {
```

```
        sc.ExecuteCommand(205);
```

```
    }
```

```
}
```

```
private void btnAltSinirAyarla_Click(object sender, System.EventArgs e)
```

```
{
```

```
    if(lblAltSinir.SelectedIndex==0)
```

```
    {
```

```
        sc.ExecuteCommand(201);
```

```
    }
```

```
    else if(lblAltSinir.SelectedIndex==1)
```

```
    {
```

```
        sc.ExecuteCommand(200);
```

```
    }
```

```
    else if(lblAltSinir.SelectedIndex==2)
```

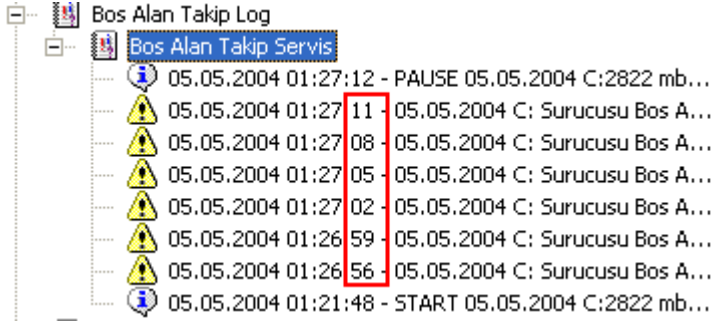
```
    {
```

```
        sc.ExecuteCommand(202);
```

```
    }
```

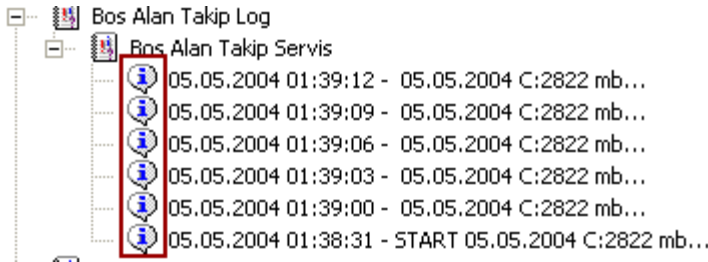
```
}
```

Şimdi uygulamamızı çalıştıralım ve ilk olarak süreyi 3 saniyede 1 seçerek Event Log'ların tutuluşunu izleyelim. Windows uygulamamızı başlatmadan önce servisimizi manuel olarak başlatmayı unutmayalım. Diğer yandan bunu dilersek SCM yardımıyla otomatik hale getirebilir ve sistem açıldığında servisin otomatik olarak başlatılmasını sağlayabiliriz. Event Log süresini 3 saniye olarak belirleyip Ayarla başlıklı butona tıkladığımızda, servisimizi temsil eden ServiceController nesnesinin ExecuteCommand metodu devreye girerek, ListBox'tan seçilen öğenin indeksine göre bir değeri, temsil ettiği servis içindeki OnCustomCommand metoduna parametre olarak gönderir. Bunun sonucu olarak Event Log'umuzda 3 saniyede bir durum bilgisinin yazılıyor olması gerekir.



Şekil 10. Servisin Event Log tutma süresinin ayarlanması.

Görüldüğü gibi servisimiz bu yeni ayarlardan sonra, kendi oluşturduğumuz Event Log'a 3 saniyede 1 bilgi yazmaktadır. Servisimizde, C sürücüsü için belli bir miktar kapasitenin altına düştüğünde, Event Log'a yazılan mesaj bir uyarı simgesi ile birlikte yazılıyor. Şu an için, belirtilen kapasitenin altında olduğumuzdan warning simgesi, servisin timer nesnesinin elapsed olayında değerlendiriliyor ve Event Log'umuza yazılıyor. Şimdi bu kapasiteyi 2 GB yapalım ve durumu gözlemleyelim.



Şekil 11. Alt Sınır değerinin değiştirilmesi.

Görüldüğü gibi, servisimizin alt sınıra göre Event Log'ların simgesini belirleyen davranışını değiştirmeyi başardık. Bu işlemi ExecuteCommand metodu ile servisimizdeki OnCustomCommand'a gönderdiğimiz parametrenin değerini ele alarak gerçekleştirdik. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde sistemimizde yer alan servisleri, ServiceController sınıfındaki metod ve özellikler ile nasıl kontrol edebileceğimizi incelemeye çalışacağız. Bu makalede görüşünceye dek hepinize mutlu günler dilerim.

Windows Servislerinin Kontrolü - 2 (Sistemdeki Servislerin Kontrol Edilmesi)

Değerli Okurlarım, Merhabalar.

Bu makalemizde, sistemde yer alan windows servislerini bir windows uygulamasından nasıl elde edebileceğimizi ve nasıl kontrol edebileceğimizi incelemeye çalışacağız. Önceki makalelerimizden hatırlayacağınız gibi, sistemde yer alan servislerimiz,

System.ServiceProcess isim alanında yer alan ServiceController sınıf nesneleri ile temsil edilmektedir. Eğer sistemde yer alan servisleri elde etmek istersek, aşağıda aşırı yüklenmiş iki prototipi olan, GetServices metodunu kullanabiliriz.

GetServices Metodu Prototipleri	Açıklaması
public static ServiceController[] GetServices();	Bu prototip ile, local makinede yer alan servisler elde edilir.
public static ServiceController[] GetServices(string);	Bu prototipte, parametre olarak verilen makine bilgisine ait servisler elde edilir.

Burada görüldüğü gibi, GetServices metodu ServiceController sınıfı türünden bir diziye geri döndürür. Ayrıca bu metod static bir metod olduğundan, doğrudan ServiceController sınıfı üzerinden herhangi bir nesne örneğine gerek duymadan çağırılabilir. Aşağıda örnek olarak sistemdeki servislerin elde edilme tekniği gösterilmektedir.

```
ServiceController[] svc;
svc=ServiceController.GetServices();
```

Elde edilen servislerin herbiri birer ServiceController nesnesi olarak, ele alınır. Elde ettiğimiz servislerin üzerinde bildiğiniz gibi, bir servisin temel davranışları olan Start,Stop,Pause,Continue,ExecuteCommand vb.larını aşağıda prototipi verilen metodlar yardımıyla gerçekleştirebilmekteyiz.

Metod	Prototip	Açıklama
Start	public void Start(); public void Start(string[]);	Servisi çalıştırır.
Stop	public void Stop();	Servisi durdurur.
Pause	public void Pause();	Çalışan servisi duraklatır.
Continue	public void Continue();	Duraklatılmış servisin çalışmasına devam etmesini sağlar.
ExecuteCommand	public void ExecuteCommand(int command);	Servisin OnCustomCommand metodunu çalıştırır.

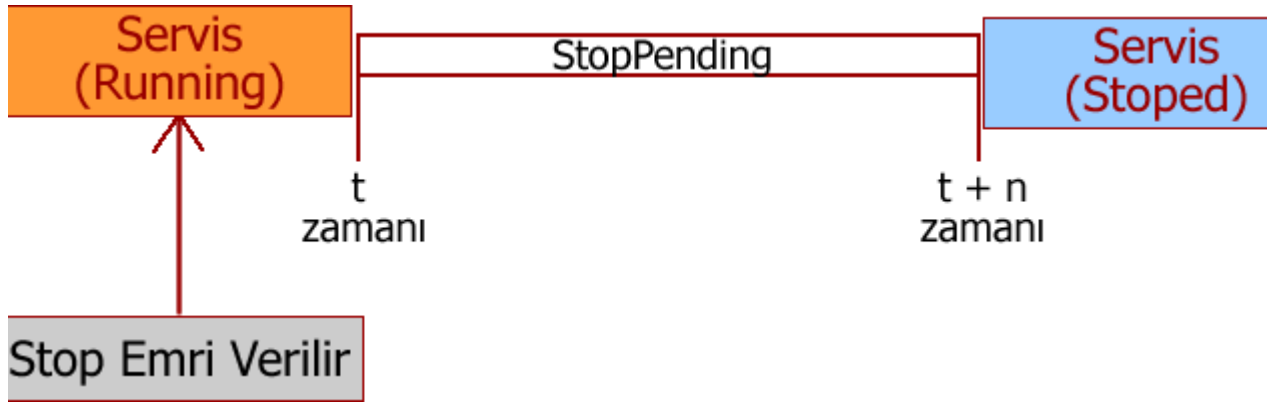
Birazdan geliştireceğimiz örnek uygulamada, sistemdeki servisler üzerinde yukarıda bahsettiğimiz metodları kullanarak, servislerin kontrolünü gerçekleştirmeye çalışacağız. Burada dikkat edilmesi gereken bir kaç nokta vardır. Öncelikle bir servis başlatılacaksa, bu servisin o an çalışmıyor olması başka bir deyişle Stopped konumunda olması gerekmektedir. Aynı durum bir servisi durdururken de geçerlidir. Böyle bir durumdada, Stop edilecek servisin, Running konumunda olması gerekmektedir. İşte bu nedenlerden dolayı, sistemdeki servislerin çalışmaya başlaması veya çalışanların durdurulması gibi hallerde, servisin o anki durumunu kontrol etmemiz gerekmektedir. Bu amaçla ServiceController sınıfının aşağıda prototipi verilen Status özelliğini kullanırız.

public ServiceControllerStatus Status {get;}

Burada görüldüğü gibi Status özelliği ServiceControllerStatus numaralandırıcısı türünden değerler alabilmektedir. Bu numaralandırıcının alacağı değerler servisin o anki durumu hakkında bilgi vermektedir. Status özelliği sadece get bloğuna sahip olduğundan, yalnızca servisin o anki durumu hakkında bilgi vermektedir. ServiceControllerStatus numaralandırıcısının alabileceği değerler aşağıdaki tabloda belirtilmektedir.

ServiceControllerStatus Değeri	Açıklama
ContinuePending	Duraksatılan servis devam ettirilmeyi beklerken.
Paused	Servis duraksatıldığında.
PausePending	Servis duraksatılmayı beklerken.
Running	Servis çalışırken.
StartPending	Servis çalıştırılmayı beklerken.
Stopped	Servis durdurulduğunda.
StopPending	Servis durdurulmayı beklerken.

Burada en çok kafa karıştırıcı haller Pending halleridir. Bunu daha iyi anlamak için örnek olarak StopPending durumunu aşağıdaki şekil üzerinden incelemeye çalışalım.



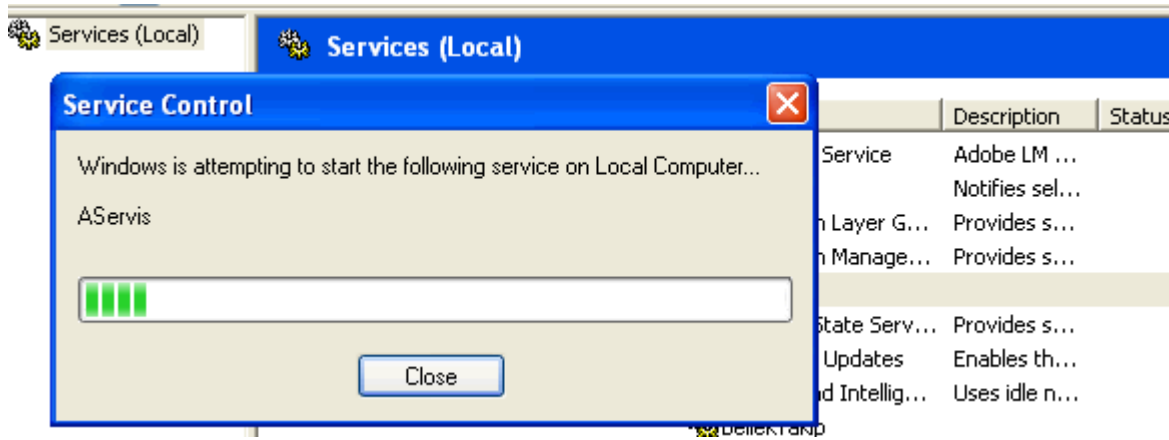
Şekil 1. Pending Durumları.

Burada görüldüğü gibi çalışan bir servise, Stop emri verildiğinde, bu servisin durumu Stopped oluncaya kadar geçen sürede, servis StopPending durumundadır. Aynı mantık, StartPending, PausePending ve ContinuePending durumları içinde geçerlidir. Pending durumlarını daha çok, bu zaman sürelerinin tamamlanıp verilen emrin başarılı bir şekilde uygulandığının izlenmesinde kullanabiliriz. Bu amaçla, ServiceController sınıfı bize, aşağıda prototipleri verilen WaitForStatus metodunu sunmaktadır. Bu metod, parametre olarak aldığı ServiceControllerStatus değeri sağlanıncaya kadar uygulamayı duraksatmaktadır.

Prototipler	Açıklama

public void WaitForStatus (ServiceControllerStatus <i>desiredStatus</i>);	Servisin ServiceControllerStatus numaralandırıcısı ile belirtilen duruma geçmesi için, uygulamayı bekletir.
public void WaitForStatus (ServiceControllerStatus <i>desiredStatus</i> , TimeSpan <i>timeout</i>);	Servisin ServiceControllerStatus numaralandırıcısı ile belirtilen duruma geçmesi için, TimeSpan ile belirtilen süre kadar uygulamanın beklemesini sağlar.

Biz uygulamamızda WaitForStatus metodunu, bir servise Start,Stop,Pause ve Continue emirlerini verdiğimizde, pending işlemlerinin sona ermesi ile birlikte servisin başarılı bir şekilde istenen konuma geçtiğini anlamak amacıyla kullanacağız. Bu aslında SCM yardımıyla bir servisin durdurulması veya başlatılmasında oluşan bekleme işlemi ile alakalı bir durumdur. Örneğin,



Şekil 2. Servisin çalıştırılması sırasındaki bekleme işlemi.

SCM yardımıyla bir servisi çalıştırdığımızda (Start) yukarıdaki gibi bir pencere ile karşılaşırız. Burada servis, StartPending konumundadır. Servis çalışmaya başladığında yani Running konumuna geldiğinde, artık StartPending konumundan çıkar. Lakin bu süre zarfında SCM uygulamasının bir süre duraksadığını görürüz. Bu duraksama servis Running halini alıncaya kadar sürer. İşte bunu sağlayan aslında WaitForStatus mekanizmasından başka bir şey değildir. Bu sayede servis Running moduna geçtiğinde, güncelenen listede servisin durumu Running olarak yazacaktır. Eğer WaitForStatus tekniği kullanılmamış olsaydı, bu durumda servis Start edildiğinde ve liste güncellendiğinde ilk aşamada Servisin durumu StartPending olacak ve ancak sonraki liste güncellemesinde Running yazacaktı. Bizde uygulamamızda WaitForStatus metodunu bu amaçla kullanacağız. Yani servis kesin olarak istediğimiz duruma ulaştığında, listemizi güncelliyeceğiz.

ServiceController sınıfı için kullanılacak diğer önemli bir özellikte CanPauseAndContinue özelliğidir. Bazı servislerin bu özellik değeri false olduğu için, bu servislerin Pause ve Continue emirlerine cevap vermesi olanaksızdır. İşte uygulamamızda bizde bir servisin bu durumunu kontrol edecek ve ona göre Pause ve Continue emirlerine izin vereceğiz. CanPauseAndContinue özelliğinin prototipi aşağıdaki gibidir.

```
public bool CanPauseAndContinue {get;}
```


ServiceController sınıfı için kullanabileceğimiz diğer önemli bir özellik ise, bir servisin bağlı olduğu diğer servislerin listesini elde etmemizi sağlayan, ServicesDependedOn özelliğidir. Prototipi aşağıdaki gibi olan bu özellik ile, bir servisin çalışması için gerekli olan diğer servislerin listesini elde edebiliriz.

public ServiceController[] ServicesDependedOn {get;}

Prototipten de görüldüğü gibi bu özellik, ServiceController sınıfı türünden bir dizi geriye döndürmektedir. Dolayısıyla bu özellik yardımıyla sistemdeki servislerin bağlı oldukları servisleride elde etme imkanına sahibiz.

Şimdi dilerseniz buraya kadar işlediğimiz yanları ile, ServiceController sınıfını kullandığımız bir örnek geliştirmeye çalışalım. Bu örneğimizde, sistemimizdeki servislerin listesini elde edeceğiz. Seçtiğimiz bir servis üzerinde Start, Stop, Pause, Continue gibi emirleri yerine getireceğiz ve bir servise bağlı olan servislerin listesine bakabileceğiz. Bu amaçla öncelikle aşağıdakine benzer bir windows uygulama formu oluşturarak işe başlayalım.

Şekil 3. Uygulamamızın Form Görüntüsü.

Şimdide program kodlarımızı oluşturalım.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.ServiceProcess; /* ServiceController sınıfını kullanabilmek için bu isim alanını ekliyoruz.*/
```

```

namespace ServisBakim
{
    public class Form1 : System.Windows.Forms.Form
    {
        .
        .
        .

        /* Sistemde yüklü olan servislerin listesini elde edeceğimiz bir metod geliştiriyoruz. Bu metodu hem servis listesini alırken
        hemde bir servis ile ilgili Start,Stop,Pause,Continue emirleri verildikten sonra, ListView kontrolündeki listeyi güncel tutmak amacıyla
        kullanacağız.*/
        private void ServisListeGuncelle()
        {
            ServiceController[] svc; /*ServiceController sınıfından bir dizi tanımlıyoruz.*/
            svc=ServiceController.GetServices(); /* Bu diziye ServiceController sınıfının static GetServices metodu yardımıyla sistemde
            yüklü olan servisleri birer ServiceController nesnesi olarak aktarıyoruz.*/

            lvServisler.Items.Clear(); /* ListView kontrolümüzün içeriğini temizliyoruz.*/
            /* Elde ettiğimiz servisler arasında gezinmek için, Servis sayısı kadar ötelenecek bir for döngüsü oluşturuyoruz. Servis
            sayısını ise, svc isimli ServiceController tipinden dizimizin Length özelliği ile elde ediyoruz.*/
            for(int i=0;i<svc.Length;i++)
            {
                lvServisler.Items.Add(svc[i].ServiceName.ToString()); /* ListView kontrolündeki ilk öğeye yani ServisAdı kısmına i indisli
                ServiceController tipinden svc nesnesinin ismini ekliyoruz. Bunun için ServiceName özelliğini kullanıyoruz.*/
                lvServisler.Items[i].SubItems.Add(svc[i].Status.ToString()); /*ListView kontrolündeki alt öğeye, servisimizin durumunu
                Status özelliği yardımıyla ekliyoruz.*/
            }
        }

        private void btnServisAl_Click(object sender, System.EventArgs e)
        {
            ServisListeGuncelle();
        }

        private void btnStart_Click(object sender, System.EventArgs e)
        {
            /* Öncelikle başlatmak istediğimiz servis bilgisini ListView kontrolünden almalıyız. Bunun için ListView kontrolünün
            FocusedItem özelliğini kullanıyoruz. Böylece kullanıcının ListView'da seçtiği servisin adını elde edebiliriz.*/
            string servis;

```

```

servis=lvServisler.FocusedItem.Text;
/*Seçilen servis adını ele alarak bir ServiceController nesnesi oluşturuyoruz.*/
ServiceController curSer=new ServiceController(servis);
/* Servis başlatma işlemini eğer seçilen servisin durumu Stopped ise gerçekleştirebilmeliyiz. Bu amaçla, önce seçilen
servise ait ServiceController nesnesinin Status özelliğine bakıyoruz. Değerin Stopped olup olmadığını denetlerken
ServiceControllerStatus numaralandırıcısını kullanıyoruz.*/
if(curSer.Status==ServiceControllerStatus.Stopped)
{
    curSer.Start(); /* Servisi çalıştırıyoruz.*/
    curSer.WaitForStatus(ServiceControllerStatus.Running); /* Servisimiz Running konumunu alıncaya dek bir süre
uygulamayı bekletiyoruz ve bu işlemin ardından servislerin listesini güncelliyoruz. Eğer WaitForStatus metodunu kullanmassak,
servis listesini güncellediğimizde, ilgili servis için StartPending konumu yazılıacaktır.*/
    ServisListeGuncelle();
}
else /* Servis zaten çalışıyorsa yani durumu(Status) Running ise bunu kullanıcıya belirtiyoruz.*/
{
    MessageBox.Show(curSer.ServiceName.ToString()+" ZATEN CALISIYOR");
}
}

private void btnStop_Click(object sender, System.EventArgs e)
{
    /* Bir servisi durdurmak için yapacağımız işlemler, başlatmak için yapacaklarımız ile aynı. Sadece bu kez, servisin Running
konumunda olup olmadığını denetliyor ve öyleyse, Stop komutunu veriyoruz.*/
    string servis;
    servis=lvServisler.FocusedItem.Text;
    ServiceController curSer=new ServiceController(servis);
    if(curSer.Status==ServiceControllerStatus.Running)
    {
        curSer.Stop(); /* Servis durduruluyor. */
        curSer.WaitForStatus(ServiceControllerStatus.Stopped);
        ServisListeGuncelle();
    }
    else
    {
        MessageBox.Show(curSer.ServiceName.ToString()+" ZATEN CALISMIYOR");
    }
}

```

```

private void btnPause_Click(object sender, System.EventArgs e)
{
    /* Bir servisi duraksatacağımız zaman, bu servisin, CanPauseAndContinue özelliğinin değeri önem kazanır. Eğer bu değer
    false ise, servis üzerinde Pause veya Continue emirlerini uygulayamayız. Bu nedenle burada ilgili servise ati CanPauseAndContinue
    özelliğinin değeri kontrol edilir ve ona göre Pause emri verilir.*/
    string servis;
    servis=lvServisler.FocusedItem.Text;
    ServiceController curSer=new ServiceController(servis);
    if(curSer.CanPauseAndContinue==true) /* CanPauseAndContinue true ise, Pause işlemini uygulama hakkına sahibiz.*/
    {
        if(curSer.Status!=ServiceControllerStatus.Paused) /* Servis eğer Paused konumunda değilse Pause emri uygulanır. Bir
        başka deyişle servis çalışır durumdaysa.*/
        {
            curSer.Pause(); /* Servis duraksatılıyor.*/
            curSer.WaitForStatus(ServiceControllerStatus.Paused); /* Servisin Paused konumuna geçmesi için bekleniyor. */
            ServisListeGuncelle();
        }
    }
    else
    {
        MessageBox.Show("SERVISIN PAUSE&CONTINUE YETKISI YOK");
    }
}

private void btnContinue_Click(object sender, System.EventArgs e)
{
    string servis;
    servis=lvServisler.FocusedItem.Text;
    ServiceController curSer=new ServiceController(servis);
    if(curSer.CanPauseAndContinue==true) /* CanPauseAndContinue true ise, Continue işlemini uygulama hakkına sahibiz.*/
    {
        if(curSer.Status==ServiceControllerStatus.Paused) /* ServiceControllerStatus eğer Paused konumunda ise Continue
        işlemi uygulanır.*/
        {
            curSer.Continue(); /* Duraksatılan servis çalışmasına devam ediyor. */
            curSer.WaitForStatus(ServiceControllerStatus.Running); /* Servisin Running konumuna geçmesi için bekleniyor. */
            ServisListeGuncelle();
        }
    }
}

```

```

else
{
    MessageBox.Show("SERVISIN PAUSE&CONTINUE YETKISI YOK");
}
}

```

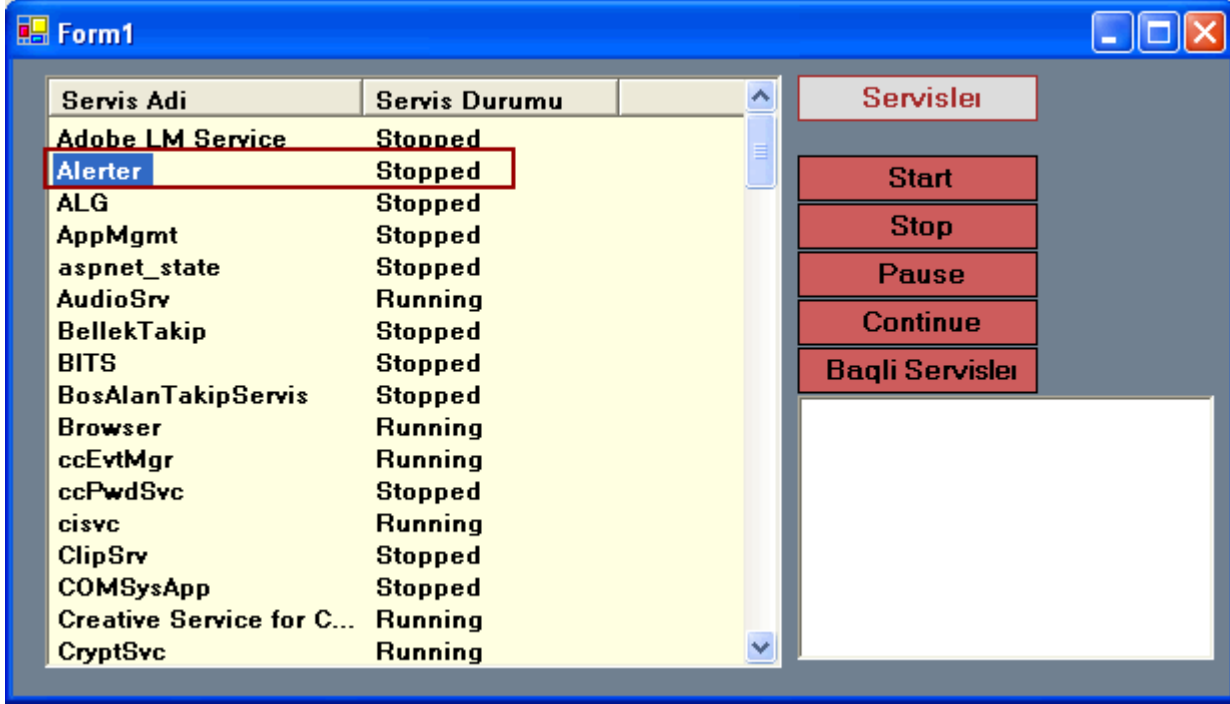
```

private void btnBagliServisler_Click(object sender, System.EventArgs e)
{
    string servis;
    servis=lvServisler.FocusedItem.Text;
    ServiceController curSer=new ServiceController(servis);
    ServiceController[] bagliServisler=curSer.ServicesDependedOn; /* Bir servise bağıli servislerin listesini elde etmek için,
güncel ServiceController nesnesinin, ServiceDependedOn özelliği kullanılır. */

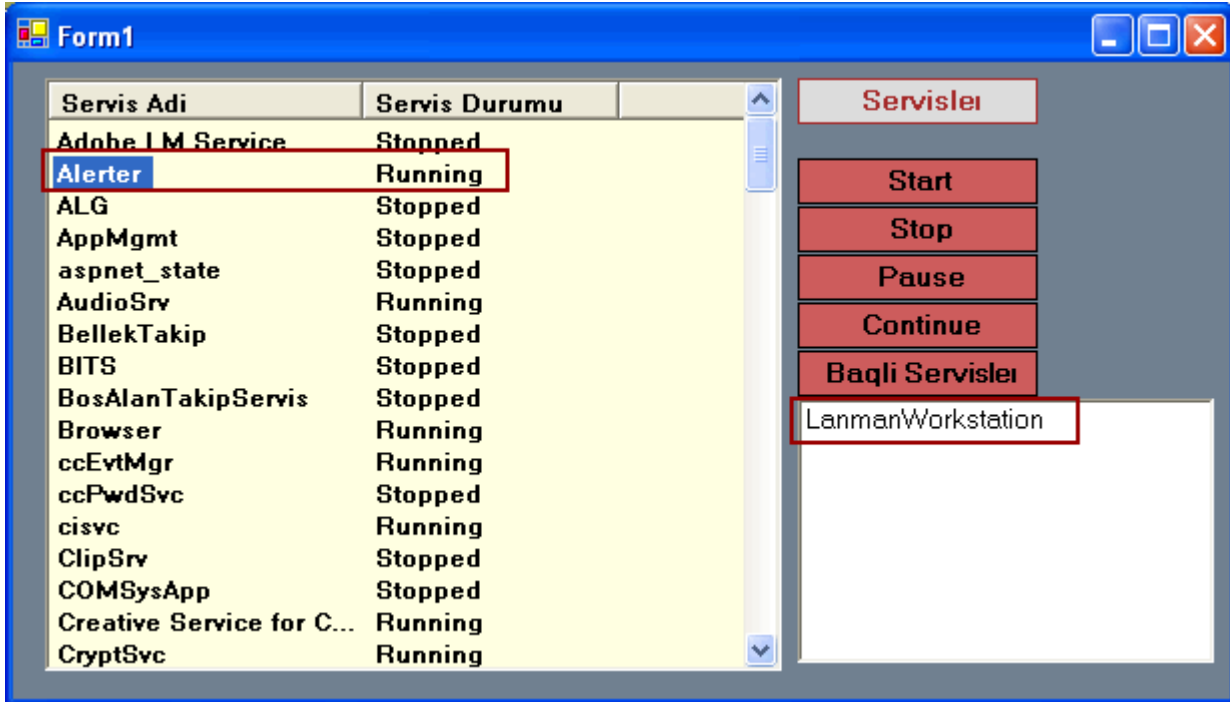
    lbBagliServisler.Items.Clear();
    /* Bağıli servisler arasında gezinmek için foreach döngüsünü kullanıyoruz.*/
    foreach(ServiceController sc in bagliServisler)
    {
        lbBagliServisler.Items.Add(sc.ServiceName.ToString());
    }
}
}

```

Uygulamamızı çalıştıralım , sistemdeki servisleri elde edelim ve örneğin Stopped konumunda olan servislerden Alerter servisini çalıştırıp, bu servise bağıli servis(ler) varsa bunların listesini tedarik edelim.



Şekil 4. Alerter Servisi Stopped konumunda.



Şekil 5. Alerter servisi çalıştırıldı ve bağlı olan servis elde edildi.

Geliştirmiş olduğumuz uygulamada oluşabilecek pek çok hata var. Örneğin listede hiç bir servis seçili değilken oluşabilecek istisnalar gibi. Bu tarz istisnaların ele alınmasını siz değerli okurlarıma bırakıyorum. Böylece geldik bir makalemizin daha sonuna. İlerleyen makalelerimizde görüşmek dileğiyle hepinize mutlu günler dilerim.

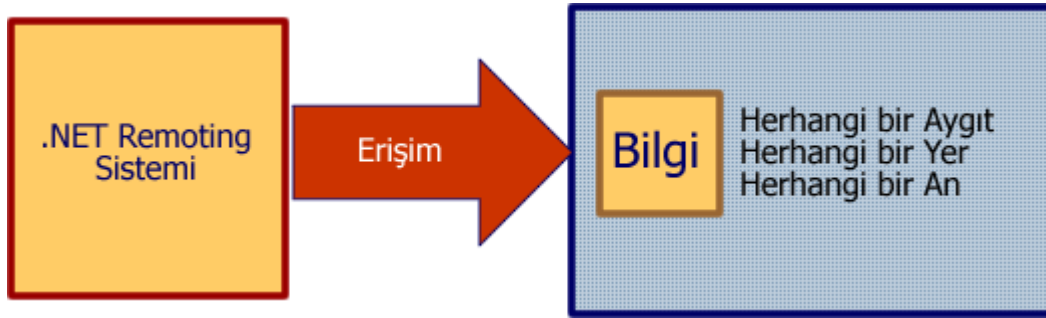
.NET Remoting'i Kavramak

Değerli Okurlarım, Merhabalar.

Bu makalemizde, .net remoting sistemini anlamaya çalışacak ve .net remoting sistemini oluşturan temel yapılar üzerinde duracağız. Öncelikle .net remoting' in ne olduğunu ve ne işe yaradığını tanımlamakla işe başlayalım. Remoting sistemi, kısaca, farklı platformlarda çalışan uygulamalar arasında veri alışverişine imkan sağlayan bir sistemdir. Bu tanımda söz konusu olan platformlar farklı işletim sistemlerinin yer aldığı farklı ve birbirlerinden habersiz proseslerde çalışan uygulamaları içerebilir. Olayın en kilit noktasıda, farklı sistemlerin veri alışverişinde bulunabilmelerinin sağlanmasıdır.

Konuyu daha iyi irdeleyebilmek amacıyla, Microsoft tarafından belirtilen şu örneği göz önüne alabiliriz. Bir pocket pc aygıtında, Windows CE işletim sistem üzerinde, C# dili ile yazılmış bir uygulamamız olduğunu düşünelim. Bu uygulama herhangi bir parçası olmadığı bir ağda yer alan bir uygulamanın ilgili metodlarını çağırıyor olsun. Uzak network üzerinde kurulu olan bu uygulama, Windows 2000 işletim sisteminde çalışan VB ile yazılmış ve başka bir sql sunucusunda yer alan bir takım bilgileri tedarik eden bir yapıya sahip olabilir. İşte .net remoting ile bu iki farklı uzak nesne arasında kolayca iletişim kurabilir ve veri alışverişini sağlayabiliriz.

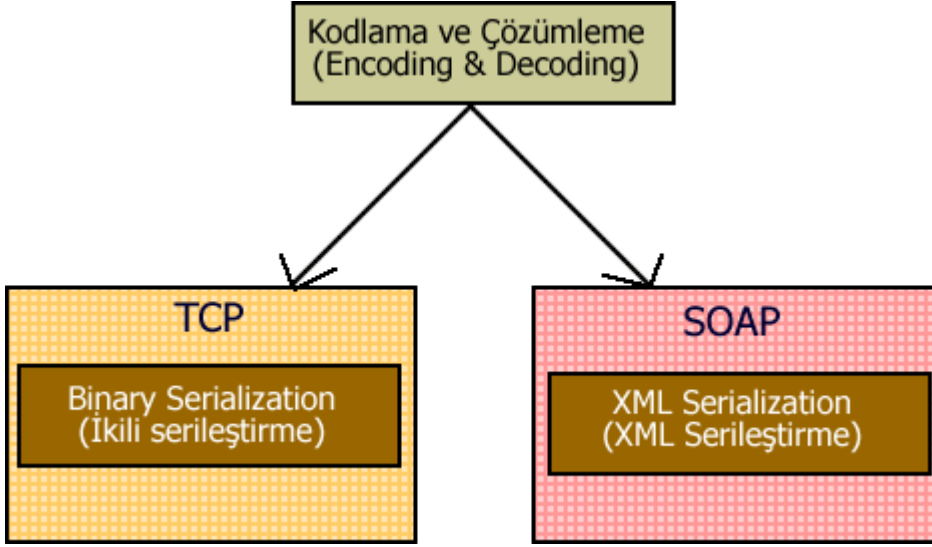
Remoting sisteminin bize vadettiği aslında, herhangi bir zamanda, yerde ve aygıt üzerinden bilgi erişimine izin verilebilmesidir. Bu aşağıdaki şekilde daha anlaşılır bir biçimde ifade edilmeye çalışılmıştır.



Şekil 1. NET Remoting Sisteminde Temel Amaç

Gelelim .net remoting sisteminin temel olarak nasıl çalıştığına ve nelere ihtiyaç duyduğuna. Öncelikli olarak sistemde uzak sınırlarda yer alan nesnelerin olduğunu söyleyebiliriz. Bu nesneler arasında meydana getirilecek iletişim, .net remoting alt yapısında yer alan bir takım üyeler ile sağlanacak. Herşeyden önce, uzak nesneler arasında hareket edecek mesajların taşınması işlemi ile, iletişim kanallarının (Communication Channels) ilgilendiğini söyleyebiliriz. Uzak nesneler arasındaki tüm mesajlar bu kanal nesneleri yardımıyla taşınacak, kodlanacak (encoding) ve çözülecektir (decoding) . Kodlamaktan kastımız, uzak nesneler arasında taşınacak mesajların, kullanılan iletişim protokolünün tipine göre belli bir formatta serileştirilmesi (serialization)dir. Diğer yandan kodlanan bu mesajın aynı kurallar çerçevesinde diğer uzak nesne tarafından çözümlenmesi (decoding) gerekecektir. Zaten aradaki mesajların yaygın iletişim protokolleri üzerinden gerçekleştirilmesi, remoting'in en temel özelliklerinden birisidir.

Bir iletişim kanalı nesnesi yardımıyla taşınan mesajlar, doğal .net serileştirici formatları (binary, soap) kullanılarak, kodlanır ve çözülürler. Bir uzak nesne, başka bir uzak nesneye mesaj gönderdiğinde, bu mesaj, kullanılan kanal nesnelerinin esas aldığı protokoller çerçevesinde serileştirilerek binary yada xml formatına dönüştürülür. Mesajı alan uzak nesne ise, serileştirilmiş bu mesajı uygun protokol tabanlı .net serileştirme formatlarını kullanarak açar ve kullanır. Serileştirme amacıyla kullanılan iki kodlama çeşidi vardır.

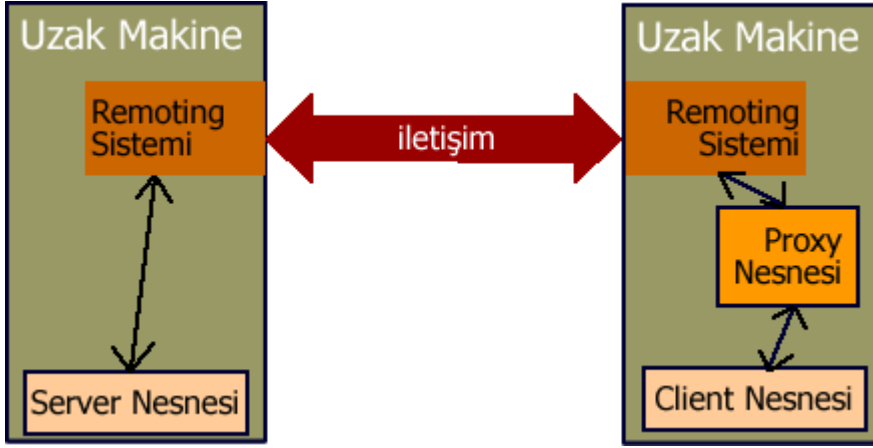


Şekil 2. Serileştirme Seçenekleri

Gelelim diğer önemli bir konuya. Uzak nesneler nasıl kullanılacak. Özellikle server (Sunucu) nitelikli merkezi nesnelere, client (istemci) uygulamalardan nasıl erişilebilecek. Bunun için uzak nesnelere ait referanslara ihtiyacımız olacaktır. Ancak burada karşımıza adresleme sorunu çıkacaktır. Nitekim, herhangi bir proses içinde çalışan bir nesne için, uzak nesnelerin çalıştığı prosesler anlamlı değildir. Bu sorunun aşılmasında kullanılabilecek ilk yol sunucu nesnesinin, istemcilere kopyalanmasıdır. Bu durumda istemci uygulama, sunucu nesnesinin bir örneğine ihtiyaç duyduğunda bunu local olarak kullanabilecektir.

Ancak kopyalama yönteminin bir takım dezavantajları vardır. Herşeyden önce çok sayıda metod içeren büyük boyutlu nesnelerin kopyalanması verimlilik ve performans açısından negatif bir etki yaratacaktır. Bununla birlikte, istemci uygulamalar, server nesnelerinin sadece belirli metodlarını kullanıyor olabilirler. Buda nesnenin tamamının kopyalanmasının anlamsız olmasına neden olan bir diğer durumdur. Bir diğer dezavantaj ise, server nesnesinin bulunduğu sistemdeki fiziki adreslere referanslar içerebilecek olmasıdır. Örneğin dosya sistemine başvuruda bulunan nesneleri barındıran server nesneleri söz konusu olabilir. Böyle bir durumda elbetteki kopyalama sonucu bu referanslar yitirilecek ve istemci tarafından kullanılamıyacaktır. Son olarak, server nesnesinin kopyalanmasının, istemci kaynaklarını harcadığını söyleyebiliriz.

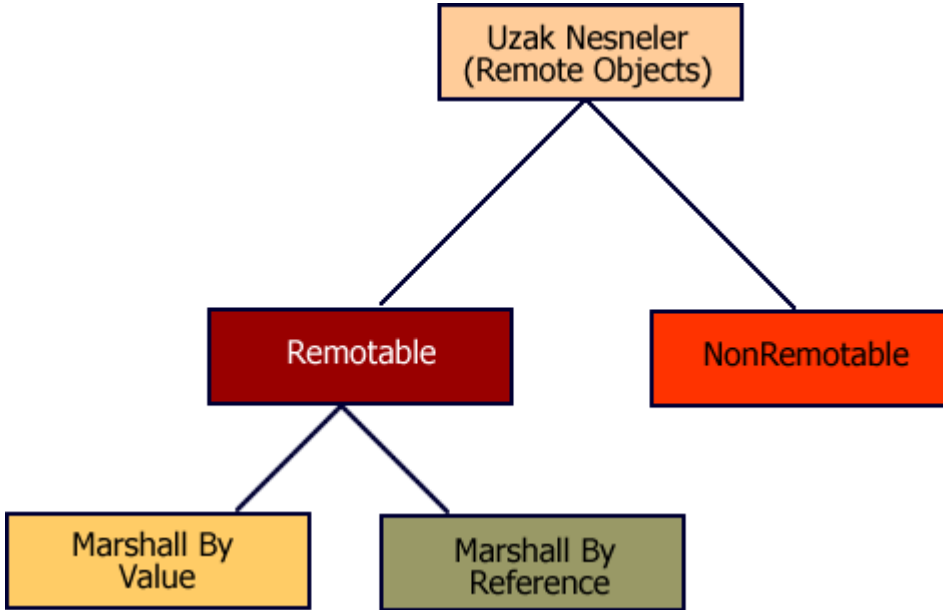
Bu dezavantajlar göz önüne alındığında bize başka bir yöntem gerekmektedir. Bu teknikte, server nesnelerini kullanmak için, bu nesnelere referansta bulunan proxy nesneleri kullanılır. İstemci uygulama, bir uzak nesneye ihtiyaç duyduğunda, bu talebini proxy nesnesine iletecektir. Proxy nesnesi ise, .net remoting'in sağlamış olduğu imkanlar dahilinde, ilgili uzak nesnesin ilgili üyesinin referansına başvuracak, bu metodu çalıştıracak ve dönen sonuçları yine proxy nesnesi aracılığıyla, istemci uygulamaya ulaştıracaktır. Bu konuyu aşağıdaki şekil ile daha net bir biçimde zihninizde canlandırabiliriz.



Şekil 3. Proxy Nesnesinin Kullanılması

Remoting sisteminde kullanılan nesnelerin kopyalanabilmesi veya referans olarak değerlendirilmesi, remoting nesnelerinin iki temel kategoriye ayrılmasına neden olmuştur. Remotable nesneler ve NonRemotable nesneler. Remotable nesneler, Başka uygulamalarca kopyalanma veya referans tekniği ile erişilebilen nesnelerdir. NonRemotable nesneler ise, diğer uygulamalar tarafından erişilemeyen nesnelerdir. Çoğunlukla büyük boyutlu, çok sayıda metod içeren veya local olarak fiziki adres referanslarına ihtiyaç duyan uzak nesnelerin, nonRemotable olarak belirtilmesi sık görülen bir tekniktir. Peki bu durumda bu nesneleri kullanmak isteyen uzak uygulamalar nasıl bir yol izleyebilir ? İşte bu noktada, nonRemotable nesnelerin, istemcilere açılacak olan bölümleri için remotable nesneler kullanılır.

Diğer yandan remotable nesneler, kopylama veya referans taşımaya imkan veren uzak nesnelerdir. Burada ise karşımıza iki çeşit remotable nesne oluşumu çıkar. Marshall By Value tipi remotable nesneler ve Marshall by Reference tipi remotable nesneler. Bu kategorilendirme şekilsel olarak aşağıdaki gibidir.

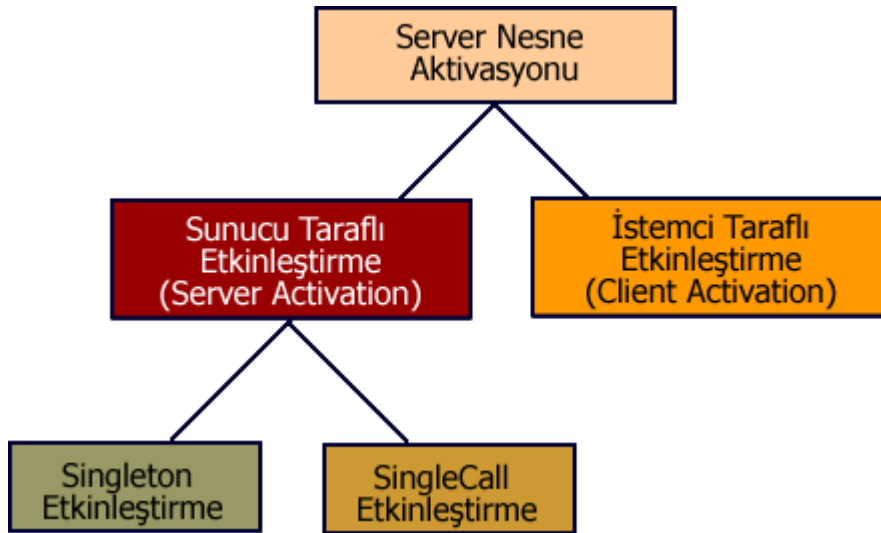


Şekil 4. Uzak Nesnelerin Temel Kategorilendirilmesi.

Özellikle remotable nesneler için yapılan ayırım kullanım açısından çok önemlidir. Marshall By Value olarak belirlenmiş remotable nesneler, bir istemci tarafından talep edildiklerinde, özellikle bu nesnenin herhangi bir metodu istemci tarafından çağırıldığında, .net remoting sistemi bu nesnenin bir kopyasını oluşturur ve bu kopyayı iletişim kanal nesneleri yardımıyla serileştirerek, çağırıcıyı yapan istemciye gönderir. İstemci tarafında yer alan, .net remoting sistemi ise bu kopyayı çözümler ve bir nesnenin bir kopyasını istemci makinede oluşturur. Nesnenin, istemciye kopyalanabilmesi için serileştirme işlemi uygulanır. Burada önemli olan nokta, serileştirmenin otomatik olarak gerçekleştirilebilmesi için, nesneye ISerializable arayüzünün uygulanmasıdır. Nesnelerin, istemcilere bu yolla taşınması özellikle istemciler açısından işlem zamanını kısaltıcı bir etken olarak karşımıza çıkmaktadır. Nitekim istemcinin talep ettiği metodlara, artık istemciye uzak nesnenin kopyası üzerinden erişilmektedir.

Marshall By Reference nesneleri ise, istemci uygulamalar için bu sistemlerde birer proxy nesnesi şeklinde oluşturulur. Bu proxy nesnesi, asıl uzak nesneye ilişkin referansları içeren metadata bilgilerine sahiptir. Uzak nesnenin herhangi bir metodu çağırıldığında, proxy nesnesi ilgili metodun referansı ile hareket eder ve bir takım bilgileri sunucuya gönderir. Sunucu gelen bilgi paketini çözümledikten sonra ilgili parametreleri değerlendirerek talep edilen metodu çalıştırır ve bunun sonucu olarak geri dönüş değerlerini istemci makinedeki proxy nesnesine gönderir. Sonuçlar istemciye bu proxy nesnesi yardımıyla gönderilecektir.

Bir uzak nesnenin herhangi bir metodunun çağırılmasında veya uzak nesneye ait bir örneğin istemciye new anahtar sözcüğü ile oluşturulmaya çalışılmasında, uzak nesnenin davranış biçimi ve aktifleştirilmesi önem kazanır. Nitekim uzak nesnenin aktif hale gelmesi iki teknik ile gerçekleştirilebilmektedir.



Şekil 5. Nesne Etkinleştirme.

Öncelikle sunucu tarafli etkinleştirmeden bahsedelim. Bu etkinleştirme tekniğinde, istemci tarafından sunucu nesnesinin bir metodu çağırıldığında, sunucu tarafından bu nesneye ait bir örnek oluşturulur. Daha sonrada bu nesne örneğinin proxy nesnesi, istemci tarafında oluşturulur. Burada dikkat çekici nokta nesne örneğinin, new anahtar sözcüğü ile değil, sunucu nesneye ait bir metod çağırıldığında oluşturuluyor olmasıdır.

Bu etkinleştirme tekniğine örnek olarak şunu gösterebiliriz. Devlet dairelerindeki sunuculara bağlı bürolar olduğunu düşünelim. Sunucu nesnemiz, bu bürolara, vergi numarasına göre kontrol ve kimlik bilgisi değerlerini gönderiyor olsun. Bu örnekte büro istemcilerinin, sunucuya sürekli bağlı olduklarını düşünelim. İstemci, bir vergi numarasını kontrol etmek istediğinde, sunucu

nesnesindeki ilgili metodu çağıracaktır. İşte bu noktada sunucu taraflı etkinleştirme devreye girerek, metod çağırımına karşılık sunucu nesnesinin örneğini oluşturur.

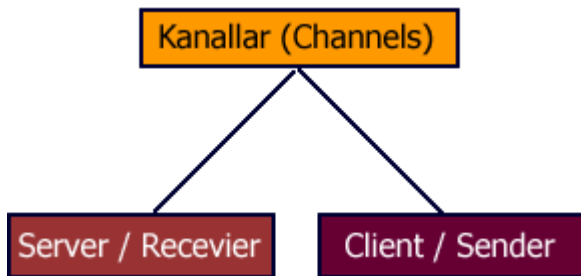
Diğer yandan sunucu taraflı etkinleştirmede, Singleton ve SingleCall teknikleri kullanılmaktadır. Singleton tekniğine göre etkinleştirmede, sunucu nesneyi kullanan kaç istemci olursa olsun, her bir istemcinin metod çağırımları sunucu tarafındaki tek bir nesne örneği tarafından yönetilmektedir. SingleCall tekniğinde ise, istemci tarafından yapılan her metod çağırısına karşılık birer sunucu nesnesi örneği oluşturulacaktır.

İstemci taraflı etkinleştirmeye gelince. Bu kez sunucu taraflı etkinleştirmenin aksine, new anahtar sözcüğü ile bir sunucu nesne örneği istemci uygulamada oluşturulduğunda, sunucu üzerinde sunucu nesnesinin örneği oluşturulacaktır. Bu tip kullanıma örnek olarak chat uygulamalarını gösterebiliriz.

Remoting sisteminde uzak nesneler dışında en önemli unsur mesajları taşıyan kanal nesneleridir. (Channels) Kanal nesneleri uzak bilgisayarlarda yer alan uzak nesneler arasındaki haberleşmede rol oynayan kilit nesnelerdir. Aradaki iletişimi sağlamak için kullanılan kanal nesneleri, bu iletişim üzerinden mesajların gönderilmesi, taşınması ve alınması gibi işlemlerden sorumludurlar. Bildiğiniz gibi kanal nesnelerinin taşıdığı mesajlar HTTP veya TCP protokolleri çerçevesinde hareket ederler. Bir kanal nesnesi yardımıyla, uzak kanallara mesaj gönderebilir yada uzak kanallardan gelen çağrıları dinleyebiliriz.

Bir uzak nesne, başka bir uzak nesneye mesaj gönderdiğinde, kanal nesneleri devreye girerek bu mesajı binary veya xml formatında serileştirir. Mesajı yani çağrıyı alan uzak nesne ise, bu mesajın içeriğini, buradaki kanalın mesajı binary veya xml formattan çözümlemesi ile okuyabilir. Remoting sisteminde kullanılan kanallara ilişkin sınıflar ve arayüzler System.Runtime.Remoting.Channels isim alanında yer almaktadır.

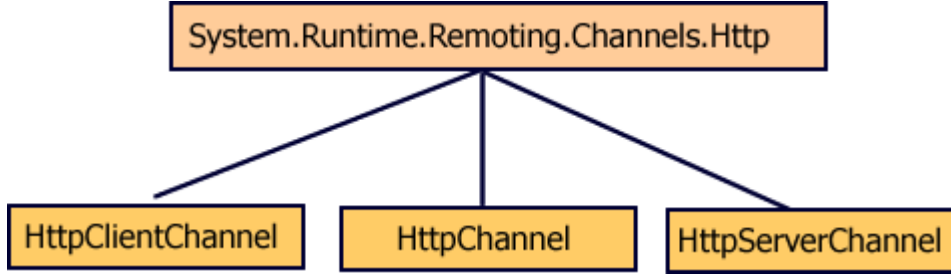
Temelde bütün kanal nesneleri IChannels arayüzünü uygulamaktadır. Kanal nesneleri iletişimde oynayacakları role göre kategorilendirilirler. Eğer kanal nesnesi çağrıları dinlemek amacıyla kullanılacaksa, alıcı (receiver) veya server (sunucu) olarak adlandırılırlar. Bununla birlikte, mesaj göndericek olan kanal nesneleri sender (gönderici) veya client (istemci) olarak adlandırılırlar.



Şekil 6. Kanallar.

Receiver kanallar, IChannelSender arayüzünü uygulayan kanallardır. Kullandıkları protokollere göre HttpClientChannel ve TcpClientChannel sınıflarından oluşturulurlar. Diğer yandan Sender kanallar, IChannelReceiver arayüzünü uygulayan TcpServerChannel ve HttpServerChannel nesneleridir. Diğer önemli iki kanal nesnesi ise HttpChannel ve TcpChannel nesneleridir.

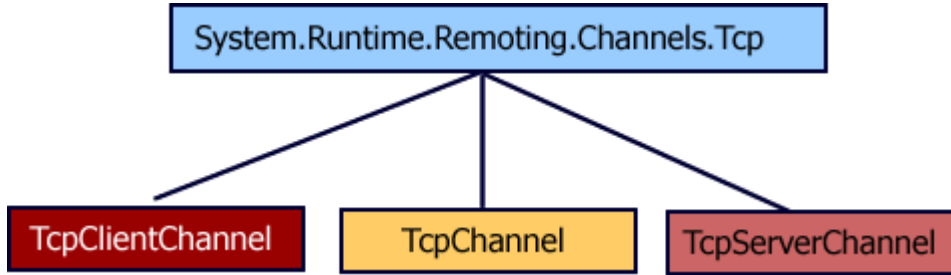
Şimdi dilerseniz bu kanalların .net remoting sisteminde oynadıkları rolleri kısaca incelemeye çalışalım. Burada önemli olan, mesajlaşmanın gerçekleştirileceği protokoldür. Nitekim HTTP protokolünü kullanacaksa buna uygun kanal nesnelerini kullanmalı, TCP protokolünü kullanacaksa buna uygun kanal nesnelerini kullanmalıyız. Eğer HTTP protokolü kullanılacaksa, System.Runtime.Remoting.Channels.Http isim alanında yer alan kanal sınıflarını kullanırız.



Şekil 7. Http Kanalları.

İstemciden, uzak nesnelere mesaj göndermek için HttpClientChannel sınıfına ait nesne örnekleri kullanılır. Diğer taraftan, istemcilerden gelecek çağrılar dinleyecek kanal nesneleri ise, HttpServerChannel sınıfından örneklenir. HttpClient sınıfına ait nesne örnekleri ise mesaj almak ve mesaj göndermek için kullanılırlar. HttpClientChannel nesne örnekleri oluşturulurken, bu kanalın kullanacağı bir port numarasını özellikle belirtmemiz gerekmez. Remoting sistemi o an için kullanılabilen serbest portlardan birisini bu kanal nesnesi için tahsis edecektir. Diğer yandan çağrılar dinlemek amacıyla tanımlanan bir HttpServerChannel kanal nesnesinin belirli bir port numarası üzerinden oluşturulması gerekmektedir. Şayet o an için kullanımda olan bir port belirlenirse çalışma zamanında istisna alırız. Bununla birlikte HttpServerChannel nesnesinin yapıcı metoduna 0 değerini göndererek, port seçme insiyatifini otomatik olarak .net remoting sistemine bırakabiliriz.

Http kanalları SoapFormatter sınıfını kullanarak, mesajları xml formatında serileştirirler. Bununla birlikte, TCP protokolünü kullanan kanal nesneleri ise, serileştirme işlemini binary olarak yapar ve bunun içinde BinaryFormatter sınıfını kullanırlar. TCP protokolünü taban alan kanal nesneleri, System.Runtime.Remoting.Channels.Tcp isim alanında yer alan sınıflardan örneklenirler.



Şekil 8. Tcp Kanalları.

Tcp isim alanında yer alan sınıflar, Http'dekiler ile aynı işlevlere sahiptirler. Tek fark, kullanılan serileştirme işleminin farklı oluşudur. Her iki isim alanı içinde, oluşturulan kanal nesne örneklerinin ChannelServices sınıfında yer alan static RegisterChannel metodu ile sisteme kayıt edilmeleri gerekmektedir.(Registration)

Kanalların kullanımında dikkat edilmesi gereken en önemli nokta, uzak nesnelerin aynı protokolü destekleyen kanalları kullanmalarının gerekli olduğudur. Örneğin, istemcilerden gelen çağrılar dinlemek amacıyla Http protokolünü taban alan kanal nesneleri kullanılıyorsa, istemcilerde de mesaj göndermek için Http protokolünü taban alan kanal nesneleri kullanılmalıdır. Nitekim farklı protokol tabanlı kanalların kullanılmasında istisnalar alırız. Bunun sebebi, farklı protokol kullanımının sonucu olarak kodlama ve çözümleme işlemlerinin farklı serileştirme teknikleri içerisinde yapılıyor olmasıdır. Http ve Tcp kanalları arasındaki farkları şu şekilde özetleyebiliriz.

1	Http kanal nesneleri Http protokolünü, Tcp kanal nesneleri ise Tcp protokolünü kullanır.
2	Serileştirme işleminde, Http kanal nesneleri SoapFormatter sınıfını kullanırken, Tcp kanal nesneleri BinaryFormatter sınıfını kullanır.
3	Http kanal nesneleri için serileştirme xml tabanlı yapılırken, Tcp kanal nesneleri için serileştirme binary formatta yapılır.

Bu makalemiz ile .net remoting sisteminin temel yapıtaşlarını tanımaya çalıştık. Bir sonraki makalemizde, en basit anlamda bir remoting uygulamasının nasıl yapıldığını incelemeye çalışacak ve teoride anlattıklarımızın programatik olarak nasıl yazılacağını göreceğiz. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

XML Rapor Web Servisleri

Değerli Okurlarım, Merhabalar.

Bu makalemizde, Crystal Report'ların birer web servisi olarak nasıl yayınlanacaklarını ve istemciler tarafından kullanılacaklarını kısaca incelemeye çalışacağız. Hepimizin bildiği gibi Web Servislerinin günümüz teknolojilerine getirdiği en büyük yenilik, merkezileştirilmiş metodların, herhangi bir platformda yer alan sayısız istemci tarafından, hiç bir engelle yada kısıtlamaya takılmadan kolayca çağırılabilmesi ve sonuçların aynı yollar ile kolayca sorunsuz elde edilebilmeleridir. Öyleki, web servislerinin XML tabanlı olarak, SOAP protokolünün belirlediği kriterlerde HTTP gibi basit iletişim protokolleri üzerinden anlaşmayı desteklemesi, onların esnek, genişleyebilir, kolay erişilebilir ve popüler olmalarını sağlamıştır.

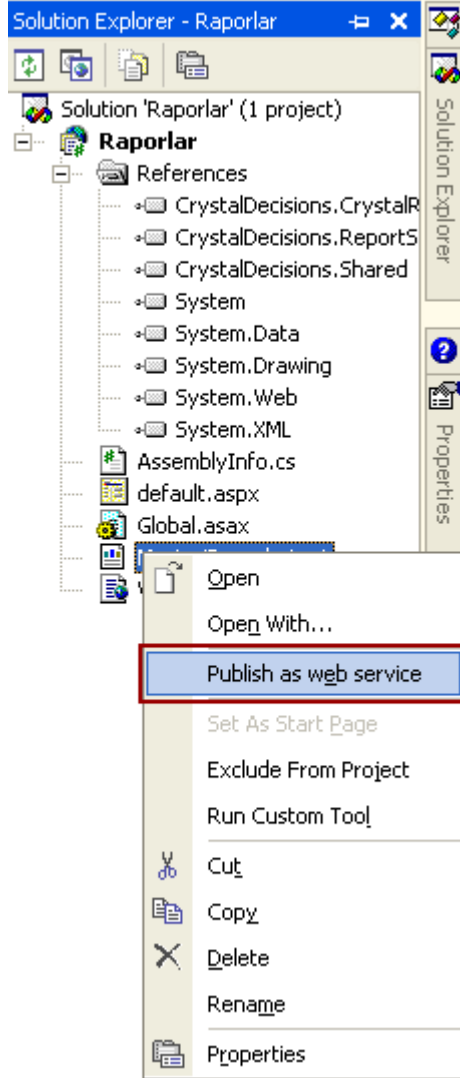
İşte web servislerinin sunmuş olduğu bu imkanlardan, geliştirmiş olduğumuz Crystal Report'larında faydalanmalarını sağlayabiliriz. Çoğunlukla, müşterilerimizin yada firmamız ile birlikte çalışan tedarikçilerimizin, ortak bir noktadan, Crystal Report olarak hazırladığımız esnek ve güçlü raporlara, her platformdan sorunsuz ulaşılabilmesi amacıyla web servisi tabanlı raporlar geliştirilir.

XML Rapor Web Servislerinin geliştirilmesi iki aşamalı bir işlem den ibarettir. İlk olarak, rapor dosyamızı baz alacak bir web servisinin geliştirilmesi gerekir. Bu raporu kullanmak için ise, standart bir web servisinin kullanılmasında uygulanan prosedürler işletilir. Yani, web servisine ait referans, uygulamaya eklenir. Sonrasında ise, eğer uygulamayı Visual Studio.Net ile geliştirdiysek, web servisimiz için uygulamamızda bir proxy nesnesi, bu servisin istemci uygulamaya indirilen wsdl dökümanı vasıtasıyla otomatik olarak oluşturulur. Sonrasında, istemci uygulamamız ile web servisimiz arasındaki mesajlaşmalar SOAP protokolünün belirlediği tipte bu proxy nesnesi yardımıyla gerçekleştirilir. Dolayısıyla, geliştireceğimiz istemci uygulamada, bu proxy sınıfına ait nesne örneğini kullanmamız gerekecektir.

Lafı fazla uzatmadan XML Rapor Web Servislerinin geliştirilmesi ile işe başlayalım. Bu amaçla ilk olarak Visual Studio.Net ortamında bir Asp.Net Web Uygulaması açalım. Bu noktada uygulamamıza, Solution Explorer'dan sağ tıklayarak açtığımız menüden Add Existing Item ile daha önceden geliştirmiş olduğumuz herhangi bir rpt uzantılı rapor dosyasını ilave edelim. (Bu noktada asıl amacımız rapor dosyasının nasıl oluşturulduğu olmadığı için, bu konunun üstünde fazla durmayacağım.)

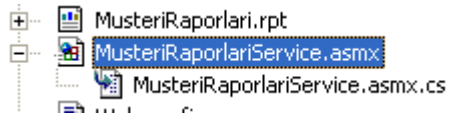
Ben uygulamamda örnek olarak, daha önceden geliştirdiğim çok basit bir rapor dosyasını kullandım. Bu rapor, Sql Server sunucusunda yer alan Northwind veritabanındaki Customers tablosuna ait bir kaç veriden oluşmakta. Bizim için asıl önemli olan

kısım, bu raporun, bir web servisi olarak yayınlanmak istenmesidir. İşte bu amaçla raporumuza sağ tıklayıyor ve çıkan menüden Publish As Web Service seçeneğini işaretliyoruz.



Şekil 1: Raporun Web Servisi Olarak Yayınlanması.

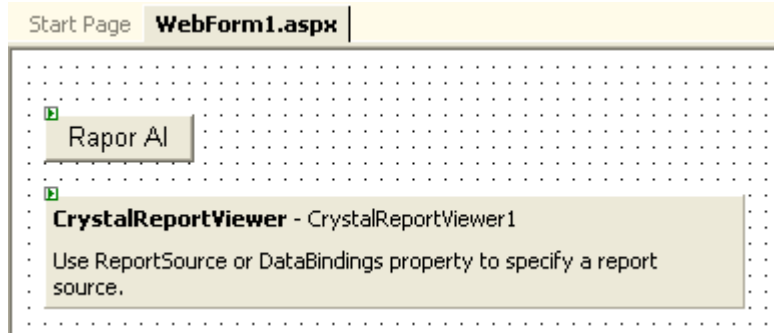
Bu işlemin ardından raporumuz için, asmx uzantılı web servisi dosyamızın oluşturulduğunu görürüz.



Şekil 2: Raporumuzun Web Servisi Haliye Getirilmesi.

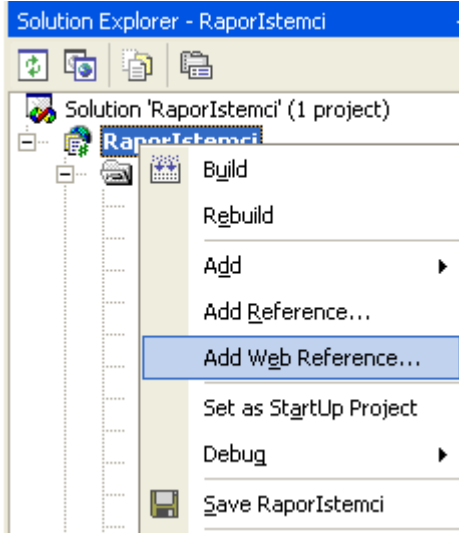
Dilersek bu sayfayı web browser'dan çalıştırarak raporumuz ile ilgili olarak neler yapabileceğimizi görebiliriz. Artık elimizde, raporumuzu kullanmak isteyen istemcilere sunabileceğimiz bir web servisimiz var. Son adım olarak uygulamamızı derleyelim. Şimdi bu XML Rapor Web Servisini kullanmak isteyen istemcileri nasıl geliştireceğimize bakalım. Bunun için örnek olarak bir Asp.Net web

uygulaması açalım. Bu uygulamaya web servisinden elde edeceğimiz raporu gösterecek olan bir Crystal Report Viewer nesnesini ekleyelim. Sayfamız aşağıdakine benzer bir yapıda olabilir.



Şekil 3: İstemci Asp.Net sayfasının tasarım görünümü.

Tanımlanmış olan bir web servisini, bir istemciye eklemek için, Add Web Reference seçeneğini kullanırız. Bu durum, elbetteki XML Rapor Web Servisleri içinde geçerlidir.



Şekil 4. XML Rapor Web Servisinin İstemci Uygulamaya Referans Edilmesi.

Buradab karşımıza aşağıdaki pencere gelicektir. Biz yerel makinede çalıştığımız ve web servisimizde localhost'umuzda geliştirdiğimiz için, kullanılabilir web servislerini öğrenmek amacıyla getirilen listeden, Web services on the local machine linkini seçebiliriz.

Start Browsing for Web Services

Use this page as a starting point to find Web services. You can click the links below, or type a known URL into the address bar.

Browse to:

- [Web services on the local machine](#)
- [Browse UDDI Servers on the local network](#)
Query your local network for UDDI servers.
- [UDDI Directory](#)
Query the UDDI business registry to find companies and production Web services.
- [Test Microsoft UDDI Directory](#)
Locate test Web services to use during development.

Şekil 5. Kullanılabilir web servislerinin öğrenilmesi.

Bu linki seçtiğimiz takdirde sistemde kullanabileceğimiz web servisleri ekrana gelecektir. Bu servislerden bir tanesinde, geliştirmiş olduğumuz XML Rapor Web Servisidir. Bu raporu seçerek, web servisine ait referansı uygulamamıza ekleyelim.

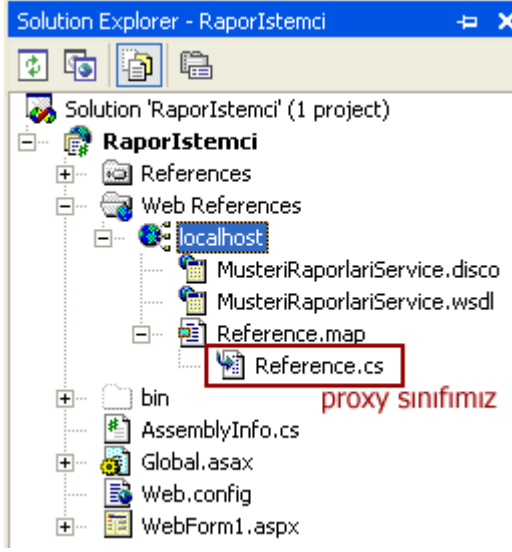
Web Services on the Local Machine

The Web services and Discovery Documents available on your VS.NET developer machine are listed below. Click the service link to browse that service.

Services	URL
GeoMat	http://localhost/GeoWebServis/GeoMat.asmx
MusteriRaporlariService	http://localhost/Raporlar/MusteriRaporlariService.as

Şekil 6. XML Rapor Web Servisimizin Seçilmesi

XML Rapor Web Servisimizin seçilip, istemci uygulamamıza eklenmesi ile birlikte, vs.net bu web servisine ait wsdl dosyasını talep eder. Wsdl dökümanı web servisinin public arayüzünün bir görüntüsünü xml formatında istemci uygulamada kullanabilmemizi sağlar. Bu işlemin ardından, istemci uygulamada XML Rapor Web Servisimize ait Wsdl dökümanıda otomatik olarak oluşturulur. Bu döküman yardımıyla da, web servisimiz ile aramızdaki ilişkiyi nesnesel bazda gerçekleştirebilmemizi sağlayan proxy sınıfımız oluşturulacaktır. Bu durumda Solution Explorer' da projemizin son halinin aşağıdaki gibi olduğunu görürüz.



Şekil 7. Proxy Sınıfımız Oluşturulur.

Artık tek yapmamız gereken, XML Rapor Web Servisimizden elde edeceğimiz raporun içeriğini, Crystal Report Viewer nesnemizde göstermek. Bunun için aşağıdaki basit kod satırlarını uygulamamıza eklememiz yeterli olacaktır.

```
private void btnRaporla_Click(object sender, System.EventArgs e)
{
    /* Öncelikle proxy sınıfımıza ait bir nesne örneğini oluşturuyoruz. */
    localhost.MusteriRaporlariService rapor=new RaporIstemci.localhost.MusteriRaporlariService();
    this.CrystalReportViewer1.ReportSource=rapor; // Web sayfamızdaki CrystalReportViewer nesnemize rapor kaynağı olarak,
    proxy nesnemizi atıyoruz. */
    this.CrystalReportViewer1.DataBind(); /* Nesnemizi gelen rapor verilerine bağlıyoruz.*/
}
```

Burada olanları kısaca özetlemek gerekirse; öncelikle XML Rapor Web Servisimizin istemci uygulama için oluşturulan proxy sınıfından bir nesne örnekledik. Böylece, istemcimizdeki Crystal Rapor nesnemiz, bu proxy sınıfından elde edilen verileri gösterecek. Elbetteki proxy nesnemizde bu verileri, XML Rapor Web Servisimizden almakta. Web sayfamızı çalıştırıp, buton kontrolümüze tıkladığımızda aşağıdakine benzer bir sayfa ile karşılaşırız.

http://localhost/RaporIstemci/WebForm1.aspx



10.06.2004

<u>CompanyName</u>	<u>ContactName</u>	<u>ContactTitle</u>	<u>Phone</u>
Alfreds Futterkiste	Maria Anders	Sales Representative	030-00743
Ana Trujillo Emparedados y	Ana Trujillo	Owner	(5) 555-47
Antonio Moreno Taquería	Antonio Moreno	Owner	(5) 555-39
Around the Horn	Thomas Hardy	Sales Representative	(171) 555-
Berglunds snabbköp	Christina Berglund	Order Administrator	0921-12 34
Blauer See Delikatessen	Hanna Moos	Sales Representative	0621-0846
Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	88.60.15.3
Bóldo Comidas preparadas	Martín Sommer	Owner	(91) 555 22
Bon app'	Laurence Lebihan	Owner	91.24.45.4
Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	(604) 555-
B's Beverages	Victoria Ashworth	Sales Representative	(171) 555-
Cactus Comidas para llevar	Patricio Simpson	Sales Agent	(1) 135-55

Şekil 8. Raporun Elde Edilmesi.

Bu makalemizde en basit haliyle, XML Rapor Web Servislerinin nasıl oluşturulduğunu ve kullanıldığını incelemeye çalıştık. Görüldüğü gibi kilit nokta, raporun web servisi olarak yayınlanmasıdır. (Publish As Web Service) Diğer taraftan geri kalan bütün işlemler web servislerinin istemcilerde kullanılmasıdaki teknikler ile aynıdır. İlerleyen makalelerimizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Transaction' larda SavePoint Kullanımı

Değerli Okurlarım, Merhabalar.

Bu makalemizde, Ado.Net ile gerçekleştirilen transaction işlemlerinde, sql' de yer alan SavePoint' lerin nasıl uygulandığını incelemeye çalışacağız. Sql' de transaction işlemlerinde, her bir iş parçasından sonra gelinecek noktanın birer SavePoint olarak kaydedilmesi sık rastlanan bir tekniktir. Bir transaction birden fazla iş parçasına sahiptir. Her bir iş parçasının başarılı olması halinde, tüm bu işlemler onaylanarak (commit) kesin olarak veritabanına yansıtılır. Diğer yandan, iş parçalarının herhangi birisinde meydana gelebilecek bir aksaklık sonucu transaction RollBack işlemini uygular ve tüm işlemler yapılmamış sayılarak veritabanı, transaction başlamadan hemen önceki haline getirilir.

Ancak çoğu zaman transaction blokları içerisine aldığımız iş parçaları, çok fazla sayıda olup, herhangi bir noktada meydana gelebilecek RollBack işlemi sonucu o ana kadar yapılan tüm işlemlerin geçersiz sayılması istenen bir durum olmayabilir. İşte böyle bir durumda, başarılı bir şekilde gerçekleşen işlerden sonraki kod satırlarına dönmek daha mantıklı bir yaklaşımdır. Elbette bu durum havale, eft gibi bankacılık işlerini kapsayan transaction' larda tercih edilmemelidir.

SavePoint' lerin çalışma mantığında, transaction içindeki belirli noktaların işaretlenmesi yatmaktadır. Bu durumda, RollBack işlemi söz konusu olduğunda, bu işaretlenmiş noktalardan birisine dönülebilme imkanına sahip olunur. İşte Sql transaction' larındaki bu

tekniği, .net ile geliştirdiğimiz uygulamalarda da simule edebilmek için, Transaction sınıflarının aşağıda SqlTransaction sınıfı için prototipi verilen Save metodu kullanılır.

```
public void Save(string savePointName);
```

Bu metod parametre olarak, SavePoint' in ismini belirten string türde bir değer alır. SavePoint olarak kaydedilmiş bir noktaya RollBack işlemi ile geri dönebilmek için, RollBack metodunun aşağıdaki prototipi kullanılır.

```
public void Rollback(string transactionName);
```

Burada, RollBack metoduna, parametre olarak string türden SavePoint' in adı verilir. Burada önemli olan, RollBack ile herhangi bir SavePoint' e dönülmesinden sonra eğer Commit işlemi uygulanırsa, SavePoint' e kadar yapılan iş parçalarının kesin olarak veritabanına yansıtılacağıdır. Şimdi SavePoint kullanımına ilişkin olarak basit bir örnek geliştirelim. Örnek windows uygulmasında, 3 iş parçası içeren bir transaction kullanacağız. Bu transaction işleminde, SavePoint' lere yer verecek ve bu noktaların veritabanına olan etkisini basitçe incelemeye çalışacağız. Bu amaçla aşağıdaki form görünümüne benzer bir windows uygulması tasarlayarak işe başlayalım.

Şekil 1. Form Tasarımımız.

Şimdi uygulama kodlarımızı geliştirelim.

```

/* SqlConnection ve SqlTransaction nesnelerimizi tanımlıyoruz.*/
SqlConnection con;
SqlTransaction trans;

private void frmMain_Load(object sender, System.EventArgs e)
{
    /* Uygulamamız yüklenirken, SqlConnection nesnemizi oluşturuyoruz. Yerel sql sunucusunda yer alan, Northwind veritabanına bağlantı sağlayacağız.*/
    con = new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=SSPI");
}

private void btnBegin_Click(object sender, System.EventArgs e)
{
    /* Kullanıcı transaction'ı başlatıyor. Önce bağlantımız açılır daha sonra SqlConnection nesnemizin BeginTransaction metod ile, transaction'ımız bu bağlantı için başlatılır. Ardından açılış işlemi listBox kontrolüne yazılır.*/
    con.Open();
    trans=con.BeginTransaction();
    listBox1.Items.Add("Transaction basladi...");
}

private void btnEkle1_Click(object sender, System.EventArgs e)
{
    /*Burada transaction'ımız içinde çalıştırılacak bir iş parçası uygulanıyor. Bu iş parçasında, Personel isimli tabloya veri girişi yapılmakta.*/
    SqlCommand cmd=new SqlCommand("INSERT INTO Personel (ISIM,SOYISIM) VALUES ('BURAK','SENYURT')",con);
    cmd.Transaction=trans; /* SqlCommand için transaction nesnesi belirtilir.*/
    try
    {
        int sonuc=cmd.ExecuteNonQuery(); /* Komutumuz çalıştırılır.*/
        trans.Save("Insert_1"); /*Transaction'ımız içinde bu noktada bir SavePoint oluşturulur.*/
        listBox1.Items.Add(sonuc+" Kayit eklendi. SavePoint=Insert_1"); /* Durum listBox'a yazılır.*/
    }
    catch
    {
        listBox1.Items.Add("HATA...");
        trans.Rollback(); /* Bir hata oluşursa ilk hale dönülür.*/
    }
}

```

```
private void btnEkle2_Click(object sender, System.EventArgs e)
{
```

/*Burada transaction' ımız içinde çalıştırılacak bir iş parçası uygulanıyor. Bu iş parçasında, Per isimli tabloya veri girişi yapılmakta. Ancak tablomuzun ismi Personel olmalı ve Per isimli bir tablomuzda yok. Bu nedenle burada bir hata oluşacaktır. İşte bu hata sonucu transaction'ımız RollBack işlemi ile Insert_1 isimli SavePoint noktasına döner.*/

```
    SqlCommand cmd=new SqlCommand("INSERT INTO Per (ISIM,SOYISIM) VALUES ('SEFER','ALGAN'),con);
    cmd.Transaction=trans;
    try
    {
        int sonuc=cmd.ExecuteNonQuery();
        trans.Save("Insert_2");/* Insert_2 isminde bir SavePoint oluşturulur.*/
        listBox1.Items.Add(sonuc+" Kayit eklendi. SavePoint=Insert_2");
    }
    catch
    {
        listBox1.Items.Add("HATA...DONUS-->Insert_1");
        trans.Rollback("Insert_1"); /* Insert_1 isimli SavePoint noktasına dönülür.*/
    }
}
```

```
private void btnSil_Click(object sender, System.EventArgs e)
{
```

/*Personel tablosundan ID alanının değeri 1 olan satırı silecek komutu içeren bir iş parçası uygulanıyor. Ancak ID=1 olan bir satır yok ise bir istisna oluşacaktır. Bu durumda RollBack işlemi ile, Insert_2 isimli SavePoint noktasına dönülür. */

```
    SqlCommand cmd=new SqlCommand("DELETE FROM Personel WHERE ID=1",con);
    cmd.Transaction=trans;
    try
    {
        int sonuc=cmd.ExecuteNonQuery();
        listBox1.Items.Add(sonuc+" Kayit silindi.");
    }
    catch
    {
        listBox1.Items.Add("HATA...DONUS-->Insert_2");
        trans.Rollback("Insert_2"); /* Insert_2 SavePoint noktasına dönülür.*/
    }
}
```

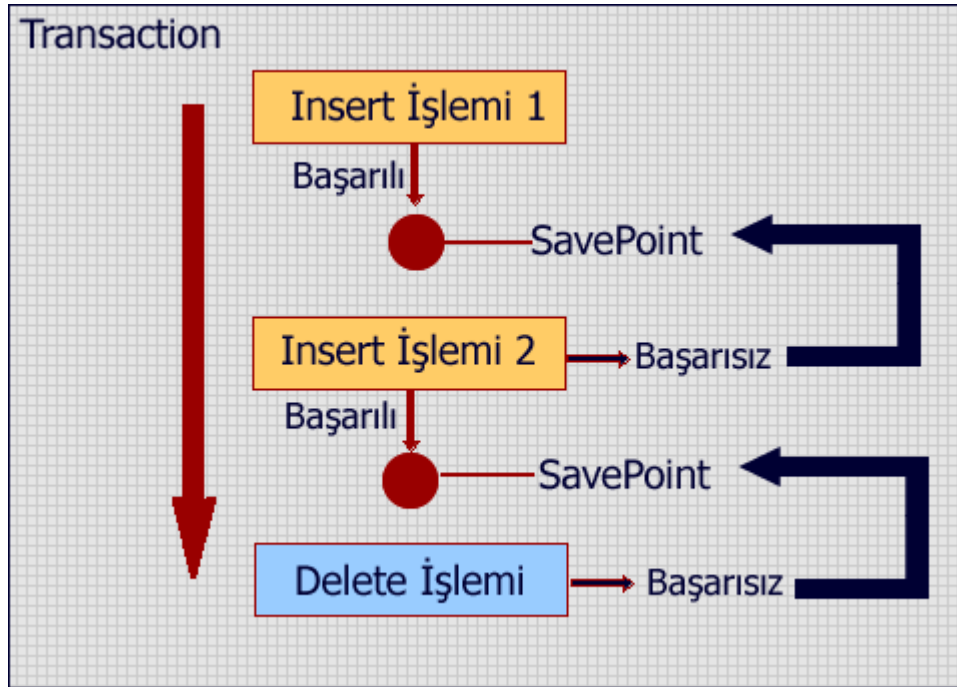
```
private void btnCommit_Click_1(object sender, System.EventArgs e)
```

```

{
    /* Transaction onaylanır. Eğer transaction herhangi bir SavePoint' te ise, o noktaya kadar olan tüm işlemler onaylanır. */
    trans.Commit();
    listBox1.Items.Add("İşlemler Onaylandı...");
    if(con.State==ConnectionState.Open)
    {
        con.Close();
    }
}

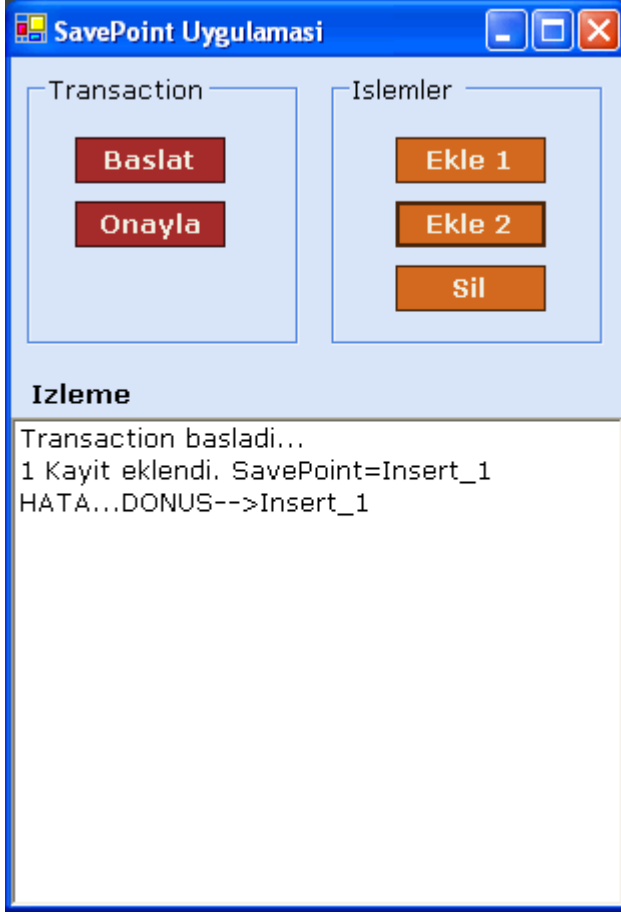
```

Burada dikkat edilecek olursa, ikinci Insert işleminde yanlış bir tablo ismi verilmiştir. Ayrıca silme işlemi için kullandığımız sql cümleciğinde, ID alanının değerinin 1 olması garanti değildir. İşte buralarda, istisnalar fırlayacaktır. Ancak catch bloklarında bu istisnalar yakandıktan sonra RollBack işlemi ile daha önceden kaydedilen belirli bir SavePoint' e dönülmektedir. İşte bu noktalardan sonra Commit işlemini uygularsak, bu SavePoint' lere kadar olan tüm iş parçaları onaylanacak ve veritabanına yansıtılacaktır. Bu durumu aşağıdaki şekil ile daha kolay anlayabiliriz.



Şekil 2. SavePoint Kullanımı

Şimdi bu durumu analiz etmek için uygulamamızı çalıştıralım.



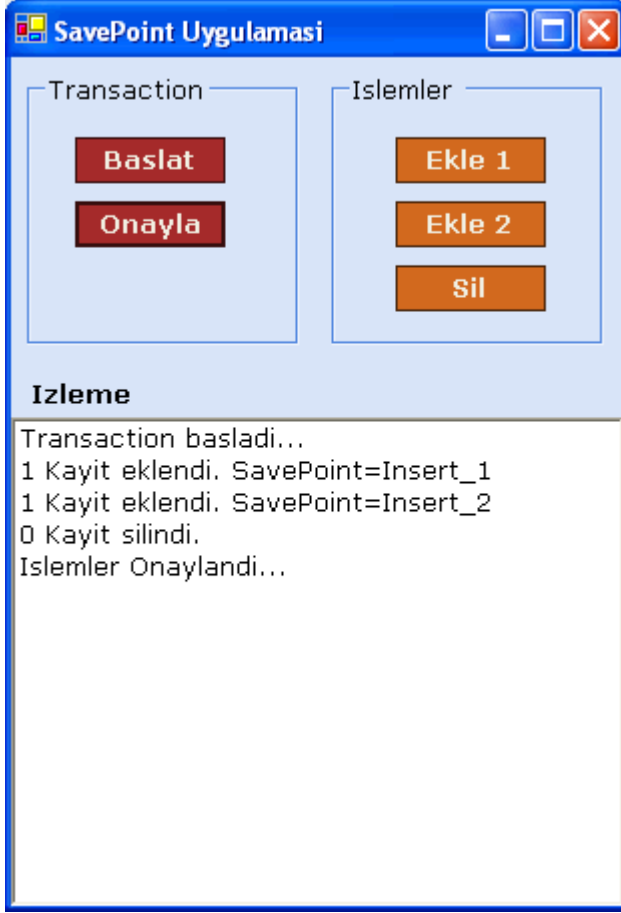
Şekil 3. Ekle 2 işleminde hata.

Önce Ekle 1 başlıklı butona basarak ilk Insert işlemimizi çalıştıralım. Daha sonra Ekle 2 başlıklı butona tıklayarak ikinci insert işlemi çalıştıralım. Burada Per isimli tablo olmadığı için bir istisna oluşacaktır. Bu durumda Catch bloğundaki RollBack metodu ile transaction Insert_1 isimli SavePoint'e döner. Eğer bu anda Onayla başlıklı butona tıklar ve transaction'ı onaylarsak (Commit) tablomuzun görünümünün aşağıdaki gibi olduğunu farkederiz. (Tablonun ilk hali hiç bir kayıt içermemektedir.)

ID	ISIM	SOYISIM
51	BURAK	SENYURT

Şekil 4. Insert_1 SavePoint'inden Sonra Commit Sonucu Tablonun Durumu.

Görüldüğü gibi, Insert_1 SavePoint' ine kadar olan işlemler (ki burada sadece ilk Insert komutu oluyor) veritabanına yansıtılmıştır. Şimdi kodumuzdaki ikinci insert ifadesinde yer alan Per isimli tablo adını düzeltelim ve bunu Personel yapalım. Bu durumda ikinci Insert işlemimizde geçerli olacaktır. Ancak halen Delete işleminde problem çıkması olasıdır. Nitekim, tablomuzda yeni girdiğimiz ilk satırın ID değeri 51' dir. Identity olarak tanımlanan bu alanın 1 değerine sahip olması şu anki şartlarda imkansızdır. Dolayısıyla Delete işleminde yine bir istisna fırlayacak ve bu kez, Insert_2 isimli SavePoint' e dönülecektir. Şimdi bu son durumu analiz etmek için uygulamayı düzenleyelim ve tekrar çalıştıralım.



Şekil 5. Delete işleminde meydana gelen istisna sonrası Commit işlemi uygulanırsa.

Önce, Baslat başlıklı butona tıklayarak Transaction' ımızı başlatalım. Ardından sırasıyla Ekle 1, Ekle 2 ve Sil başlıklı butonlara tıklayalım. Bu işlemler sonucunda Delete işleminde ID'si 1 olan satır olmadığından bir istisna oluşacak ve transaction'ımız RollBack metodu ile, Insert_2 olarak isimlendirdiğimiz SavePoint' e dönecektir. Bu noktada Onayla başlıklı butona tıklarsak, Commit işlemi sonucu Insert_2 noktasına kadar olan iki Insert işlemimizde onaylanacak ve veri tablosuna yansıtılacaktır.

ID	ISIM	SOYISIM
51	BURAK	SENYURT
58	BURAK	SENYURT
59	SEFER	ALGAN

Şekil 6. Delete işlemindeki istisna sonrası Commit uygulandığında, tablonun son hali.

Bu makalemizde, Sql SavePoint' lerinin Ado.Net içindeki transaction' larda nasıl kullanıldığını incelemeye çalıştık. Geliştirdiğimiz örnek, sadece SavePoint' lerin kullanımını inceleme amacıyla olduğundan, farklı türden hatalara açıktır. Ancak önemli olan, SavePoint'lerin bir transaction nesnesi için nasıl kayıt edildikleri ve RollBack işlemleri sonucu bu noktalara nasıl dönülebildiğidir. Ayrıca, bu tip RollBack işlemleri sonrasında verilen Commit emirlerinin verileri gerçek anlamda nasıl etkilediğinede dikkat etmek gerekir. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Transaction' larda İzolasyon Seviyeleri (Isolation Level) – 1

"Felakete maruz kalmadan önce, olasılıkları iyi bilmek gerekir."

Değerli Okurlarım, Merhabalar.

Bu makalemizde, Transaction' larda kullanılan izolasyon seviyelerini incelemeye başlayacağız. İzolasyon seviyeleri, eşzamanlı olarak çalışan Transaction' ların birbirlerini nasıl etkilemesi gerektiğini belirtmekte kullanılır. Yani bir başka deyişle, bir Transaction içinde meydana gelen değişikliklerin, başka eş zamanlı Transactionlar tarafından nasıl ele alınması gerektiğini belirlememize olanak sağlar. İzolasyon seviyelerini anlamanın en iyi yolu, eş zamanlı olarak çalışan Transaction' larda meydana gelebilecek sorunları iyi anlamaktan geçer.

Eşzamanlı olarak çalışan iki Transaction göz önüne alındığında, oluşabilecek durumlar üç tanedir. Phantoms, Non-Repeatable Read, Dirty Read. Her bir durumun, çalışan Transaction' lara etkisi farklıdır. Şimdi bu problemleri incelemeye çalışalım. Bu amaçla basit bir windows uygulaması geliştireceğiz. Uygulamamız, Sql sunucusu üzerinde çalışan veritabanı üzerinde ekleme, güncelleme, veri çekme gibi işlemler yapacak. Tüm bu işlemleri birer Transaction içerisinde gerçekleştireceğiz. Sonuç olarak, aynı anda bu uygulamalardan iki proses çalıştırıp, bahsettiğimiz problemleri simüle etmeyi deneyeceğiz. Öncelikle, Visual Studio.Net ortamında bir windows uygulamasını aşağıdakine benzer forma sahip olacak şekilde oluşturalım.

Şekil 1. Form Tasarımımız.

Sıra geldi uygulama kodlarımızı yazmaya.

```
SqlConnection con;
```

```
SqlTransaction trans;
SqlDataAdapter da;

private void btnBegin_Click(object sender, System.EventArgs e)
{
    if(con.State==ConnectionState.Closed)
    {
        con.Open();
    }
    trans=con.BeginTransaction();
}

private void Form1_Load(object sender, System.EventArgs e)
{
    con=new SqlConnection("data source=localhost;database=Northwind;integrated security=SSPI");
}

private void btnCommit_Click(object sender, System.EventArgs e)
{
    trans.Commit();
}

private void btnRollBack_Click(object sender, System.EventArgs e)
{
    trans.Rollback();
}

private void btnBak_Click(object sender, System.EventArgs e)
{
    SqlCommand cmdBak=new SqlCommand("SELECT * FROM Personel",con);
    cmdBak.Transaction=trans;
    da=new SqlDataAdapter(cmdBak);
    DataTable dt=new DataTable();
    da.Fill(dt);
    dataGrid1.DataSource=dt;
}

private void btnEkle_Click(object sender, System.EventArgs e)
{
}
```

```

SqlCommand cmdGir=new SqlCommand("INSERT INTO Personel (ISIM,SOYISIM) VALUES
('"+txtIsim.Text+"','"+txtSoyisim.Text+"'),con);
cmdGir.Transaction=trans;
int sonuc=cmdGir.ExecuteNonQuery();
MessageBox.Show(sonuc+" SATIR GIRILDI");
}

```

```

private void btnGuncelle_Click(object sender, System.EventArgs e)
{
    SqlCommand cmdGuncelle=new SqlCommand("UPDATE Personel SET ISIM='"+txtIsim.Text+"',
    SOYISIM='"+txtSoyisim.Text+"' WHERE ID='"+txtID.Text,con);
    cmdGuncelle.Transaction=trans;
    int sonuc=cmdGuncelle.ExecuteNonQuery();
    MessageBox.Show(sonuc+" SATIR GUNCELLENDI");
}

```

```

private void btnBakID_Click(object sender, System.EventArgs e)
{
    SqlCommand cmd=new SqlCommand("SELECT ISIM,SOYISIM FROM Personel WHERE ID='"+txtID.Text,con);
    cmd.Transaction=trans;
    da=new SqlDataAdapter(cmd);
    DataTable dt=new DataTable();
    da.Fill(dt);
    dataGrid1.DataSource=dt;
}

```

Kodlarımızda özel olarak yaptığımız bir şey yok. Ado.Net'i kullanarak, bizim için gerekli satırları oluşturduk. Şu an için odaklanmamız gereken, eş zamanlı Transaction' lar arasındaki problemlerin irdelenmesi.

Phantoms;

İki Transaction olduğunu ve bunların eş zamanlı olarak çalıştığını düşünelim. Transaction1, veri tabanından herhangi bir tabloya ait belirli bir veri kümesini çekmiş olsun. Aynı veri kümesine Transaction2' ninde baktığını düşünelim. Transaction2, tabloya yeni satırlar girsın ve Transaction' ı onaylansın(Commit). Bu noktadan sonra, halen daha çalışan Transaction1, aynı veri kümesini tekrardan Transaction içerisinde talep ettiğinde, yeni eklenmiş satırlar olduğunu görecektir. Bu satırlar, hayalet olarak düşünülür ve bu nedelem Phantom olarak adlandırılır. Çünkü, Transaction1 işlemini sonlandırmadan veri kümesinin yeni bir şekline sahip olmuştur. Yeni eklenen satırlar onun için bir anda ortaya çıkan hayalet satırlar gibidir. Şimdi dilerseniz, bu olayı simüle etmeye çalışalım. Öncelikle, geliştirdiğimiz uygulamadan iki adet çalıştırmamız gerekiyor. Sonra bu uygulamalardan Başlat başlıklı butonlara tıklayarak Transaction' larımızı çalıştırmalıyız. Artık elimizde çalışan iki eş zamanlı Transaction var. Şimdi, yukarıda bahsettiğimiz senaryoyu uygulayalım. Önce Transaction' lardan birisi ile veri çekelim.

ID	ISIM	SOYISIM
58	BURAK	SENYURT
59	SEFER	ALGAN
*		

Şekil 2. Transaction1 veri çekiyor.

Ardından diğer uygulamada çalışan Transaction üzerinden yeni bir kaç kayıt girelim ve bu Transaction'ı Commit edelim.



Şekil 3. Eşzamanlı çalışan Transaction2 yeni satır girdi ve işlem onaylandı (Commit).

Şimdi çalışmaya devam eden yani açık olan Transaction'ın olduğu uygulamaya geçelim ve veri kümesini yeniden Bak isimli butona tıklayarak isteyelim.

Phantoms

Baslat Onayla

Bak GeriAl

Ekle

Isim Soyisim

ID Güncelle Bak

ID	ISIM	SOYISIM
70	Ahmet	Nacaroglu
58	BURAK	SENYURT
59	SEFER	ALGAN

Şekil 4. Transaction1 için, Phantom Satırlar Oluştur.

İşte burada oluşan Ahmet Nacaroglu satırı hayalet satırdır. Bu durumun meydana gelmesi için, giriş işlemlerini yapan Transaction'ın onaylanmış olması gerektiğini hatırlatmak isterim. Aksi takdirde, diğer Transaction ile veriler çekilmek istendiğinde, belirli bir süre diğer Transaction'ın bu işlemleri tamamlayıp(Commit) tamamlamıyacağı(Rollback) beklenir ve bu süre sonunda beklenen olmassa uygulama bir istisna fırlatır. (Bu nedenle, Transaction2' de girdiğimiz satırlardan sonra, Onayla başlıklı butona basmayı unutmayalım.)

Non-Repeatable Reads;

Yine iki eş zamanlı çalışan Transaction'ımız olduğunu düşünelim. Transaction' lardan birisi yine veri çekmiş olsun. Özellikle çektiği belirli bir satır olabilir. Diğer Transaction' da bu belirli satırı veya başkalarını güncellesin ve işlemi onaylansın (Commit). İşte bu noktadan sonra, diğer çalışmakta olan Transaction aynı satırlara yeniden baktığında verilerin değişmiş olduğunu görecektir. İşte bu satırlardaki ani ve bilinmeyen değişiklikler nedeni ile, bu durum Non-Repeatable Read olarak adlandırılmıştır. Şimdi bu durumu simüle edebilmek için, yine uygulamamızdan iki tane çalıştıralım ve Transaction' larımızı başlatalım. İlk Transaction ile okuduğumuz verilerden belirli bir satır üzerinde, ikinci Transaction'da değişiklik yapalım ve bu Transaction' ı onaylayalım(Commit).



Şekil 5. Transaction2' de Transaction1' de aynen görünen satırlardan birisinde güncelleme yapıldı.

Şimdi bu Transaction' ı onaylayalım ve diğer Transaction' da verileri tekrardan çekelim. Bu durumda, ID değeri 58 olan satırın verilerinin değişmiş olduğunu görülür.

The screenshot shows a Windows application window titled "Phantoms". It contains a form with several buttons and input fields. The buttons are "Baslat", "Bak", "Ekle", "Onayla", "GeriAl", "Güncelle", and "Bak". The form has input fields for "Isim", "Soyisim", and "ID". Below the form is a table with three columns: "ID", "ISIM", and "SOYISIM". The table contains three rows of data. The second row, with ID 58, ISIM Burak S., and SOYISIM SENYURT, is highlighted with a red border. There is also a red border around the "Güncelle" button.

ID	ISIM	SOYISIM
70	Ahmet	Nacaroglu
58	Burak S.	SENYURT
59	SEFER	ALGAN

Şekil 6. Transaction1 için NonRepeatable-Read durumu.

Dirty Reads;

Bu durum Phantoms ve Non-Repeatable Read durumlarına göre biraz daha farklıdır. Eş zamanlı olarak çalışan Transaction' lardan birisi, diğerinin okuduğu aynı veriler üzerinde değişiklik yapar. Aynen Non-Repeatable Read' e neden olan durumda olduğu gibi. Ancak önemli bir fark vardır. Değişiklik yapan Transaction Commit edilmeden, diğer Transaction aynı satırları tekrar okur ve değişiklikleri görür. Bu andan sonra, değişiklikleri yapan Transaction yaptığı güncellemeleri RollBack ile geri alır. İşte bu noktada, verileri okuyan Transaction' da geri alınmış (RollBack) yani veri tabanına yansıtılmamış değişiklikler halen daha var olacaktır ki aslında bu değişiklikler mevcut değildir. İşte bu durum Dirty-Reads olarak adlandırılır. Şimdi bu durumu simüle etmeye çalışalım. Ancak burada söz konusu olan Transaction' lar eş zamanlı çalışmakla birlikte, birisi Commit veya RollBack edilmeden diğerinde veri çekme gibi işlemleri yapabilmesine izin veren yapıdadırlar. O nedenle bir sonraki makalede üzerinde duracağımız IsolationLevel numaralandırıcısı türünden bir değer ile Transaction' ları çalıştırmamız lazım. Bu amaçla aşağıdaki satırı;

```
trans=con.BeginTransaction();
```

aşağıdaki ile değiştirelim.

```
trans=con.BeginTransaction(IsolationLevel.ReadUncommitted);
```

Şu aşamada bunun ne anlama geldiğinin çok önemi yok. Bunu ve diğer numaralandırıcı değerlerinin etkilerini bir sonraki makalemizde incelemeye çalışacağız. Şimdi yine iki uygulamamızı aynı anda çalıştıralım ve Transaction' larımızı başlatalım. Önce Transaction1 için verileri çekelim. Transaction2' de belli bir satır üzerinde değişiklik yapalım.



Şekil 7. Eşzamanlı çalışan Transaction2' de belirli bir satır üzerinde güncelleme yapıldı.

Şimdi Transaction1 içinde, verileri tekrardan çekelim. Bu durumda Transaction2 ile, ID değeri 70 olan satır üzerinde yapılmış olan değişikliği görürüz.

Şekil 8. Transaction1 verileri tekrardan çekiyor ve değişiklikleri görüyor.

Şimdi Transaction2' de yaptığımız güncelleme işlemi geri alalım. İşte bu durumda, Transaction1 içinde, Transaction2 tarafından satırın güncellenmiş fakat onaylanmamış, daha da önemlisi geri alınmış hali kalır. İşte bu durum Dirty Reads olarak adlandırılmaktadır. Bu üç durum ile ilgili olarak alınabilecek tedbirler, IsolationLevel numaralandırıcısı ile belirlenir. Bu numaralandırıcıyı ve kullanım şekillerini bir sonraki makalemizde incelemeye çalışacağız. Hepinize mutlu günler dilerim.

Transaction' larda İzolasyon Seviyeleri -2 (IsolationLevel Numaralandırıcısı)

Değerli Okurlarım, Merhabalar.

Bu makalemizde, Sql izolasyon seviyelerinin, .net uygulamalarında nasıl kullanıldığını incelemeye çalışacağız. Bir önceki makalemizde, izolasyon seviyeleri için söz konusu olabilecek 3 problemi ele almıştık. Bu olası problemler phantoms, non-repeatable read ve dirty read durumlarıdır. Eş zamanlı olarak çalışan Transaction' larda meydana gelebilecek bu problemleri, IsolationLevel numaralandırıcısı yardımıyla kontrol altına alabiliriz. Bu numaralandırıcının alabileceği değerler ve bu değerlerin izin verdiği(vermediği) durumlar aşağıdaki tabloda yer almaktadır.

IsolationLevel Numaralandırıcı Değeri	Olası Problemler		
	Phantoms	Non-Repeatable Read	Dirty-Read
Chaos	SqlServer tarafından desteklenmez.		
ReadCommitted	✓	✓	✗

ReadUncommitted			
RepeatableRead			
Serializable			
Unspecified	SqlServer tarafından desteklenmez.		

Görüldüğü gibi IsolationLevel numaralandırıcısının alabileceği değerler altı adettir. Chaos ve Unspecified numaralandırıcı değerleri sql server tarafından desteklenmemektedir. Bununla birlikte, bir Transaction sınıfının IsolationLevel özelliğinin varsayılan değeri ReadCommitted olarak belirtilmiştir. Bu tablonun okunuşuna gelince. Örneğin, RepeatableRead değerini ele alalım. Bir uygulamada, Transaction nesnesine izolasyon seviyesi olarak bu değer atandığı takdirde, eş zamanlı olarak çalışan Transaction' lar arasında sadece Phantoms durumuna izin verilir. Dolayısıyla, Non-repeatable Read ve Dirty-Read durumlarına izin verilmez. Burada tüm olası problemlere izin veren IsolationLevel numaralandırıcı değeri ReadUncommitted değeridir. Aksine, Serializable değeri Transaction' lar arasında doğabilecek bu problemlerin hiç birisinin olmasına izin vermez.

Şimdi derseniz, bu değerlerin aynı zamanda çalışan Transaction' lar üzerindeki etkilerini örnek uygulamamız üzerinde incelemeye çalışalım. Bu makalemizde yine bir önceki makalemizde geliştirdiğimiz uygulamayı örnek olarak kullanabiliriz. Bu kez formumuza izolasyon seviyelerini çalışma zamanında belirlememize yarayacak 4 adet RadioButton kontrolü yerleştirdik. Formumuzun görüntüsü aşağıdaki gibi olacaktır.

Elbette, gerçek uygulamalarda burada olduğu gibi izolasyon seviyelerinin çalışma zamanında bu şekilde belirlenmesi pek doğru olmayabilir. Ancak şu an için amacımız, bu izolasyon seviyelerinde aynı anda çalışan Transaction' larda nelerin olup nelerin olmadığını kontrol edebilmektir. Uygulama kodlarımız ise, aşağıda olduğu gibidir.

```
SqlConnection con;
```

```
SqlTransaction trans;
SqlDataAdapter da;

private void btnBegin_Click(object sender, System.EventArgs e)
{
    if(con.State==ConnectionState.Closed)
    {
        con.Open();
    }
    if(this.rdbReadCommitted.Checked==true)
    {
        trans=con.BeginTransaction(IsolationLevel.ReadCommitted);
    }
    else if(this.rdbReadUncommitted.Checked==true)
    {
        trans=con.BeginTransaction(IsolationLevel.ReadUncommitted);
    }
    else if(this.rdbRepeatableRead.Checked==true)
    {
        trans=con.BeginTransaction(IsolationLevel.RepeatableRead);
    }
    else if(this.rdbSerializable.Checked==true)
    {
        trans=con.BeginTransaction(IsolationLevel.Serializable);
    }

    lblDurum.Text=trans.IsolationLevel.ToString();
}

private void Form1_Load(object sender, System.EventArgs e)
{
    con=new SqlConnection("data source=localhost;database=Northwind;integrated security=SSPI");
}

private void btnCommit_Click(object sender, System.EventArgs e)
{
    trans.Commit();
}
```

```
private void btnRollBack_Click(object sender, System.EventArgs e)
{
    trans.Rollback();
}

private void btnBak_Click(object sender, System.EventArgs e)
{
    try
    {
        SqlCommand cmdBak=new SqlCommand("SELECT * FROM Personel",con);
        cmdBak.Transaction=trans;
        da=new SqlDataAdapter(cmdBak);
        DataTable dt=new DataTable();
        da.Fill(dt);
        dataGrid1.DataSource=dt;
    }
    catch(SqlException hata)
    {
        MessageBox.Show(hata.Message.ToString());
    }
}

private void btnEkle_Click(object sender, System.EventArgs e)
{
    SqlCommand cmdGir=new SqlCommand("INSERT INTO Personel (ISIM,SOYISIM) VALUES ("
+txtIsim.Text+"",""+txtSoyisim.Text+"")",con);
    cmdGir.Transaction=trans;
    int sonuc=cmdGir.ExecuteNonQuery();
    MessageBox.Show(sonuc+" SATIR GIRILDI");
}

private void btnGuncelle_Click(object sender, System.EventArgs e)
{
    SqlCommand cmdGuncelle=new SqlCommand("UPDATE Personel SET    ISIM='"+txtIsim.Text+"",
SOYISIM='"+txtSoyisim.Text+"' WHERE ID='"+txtID.Text,con);
    cmdGuncelle.Transaction=trans;
    int sonuc=cmdGuncelle.ExecuteNonQuery();
    MessageBox.Show(sonuc+" SATIR GUNCELLENDI");
}
```

```

private void btnBakID_Click(object sender, System.EventArgs e)
{
    try
    {
        SqlCommand cmd=new SqlCommand("SELECT ISIM,SOYISIM FROM Personel WHERE ID="+txtID.Text,con);
        cmd.Transaction=trans;
        da=new SqlDataAdapter(cmd);
        DataTable dt=new DataTable();
        da.Fill(dt);
        dataGrid1.DataSource=dt;
    }
    catch(SqlException hata)
    {
        MessageBox.Show(hata.Message.ToString());
    }
}

```

Kodumuzdaki en önemli nokta, başlatılacak Transaction' lar için IsolationLevel değerlerinin belirlenmesidir. Bu amaçla, SqlConnection sınıfının, BeginTransaction metodunun aşağıdaki prototipi kullanılmaktadır. Bu prototipte BeginTransaction metodu, parametre olarak IsolationLevel numaralandırıcısı türünden bir değer alır. Böylece belirtilen bağlantı için açılacak Transaction, bu parametrede belirtilen izolasyon seviyesini kullanarak çalışacaktır.

public SqlTransaction BeginTransaction(IsolationLevel iso);

Şimdi ilk olarak IsolationLevel özelliğinin varsayılan değeri olan ReadCommitted değerinden işe başlayalım. ReadCommitted değeri, Phantoms ve Non-Repeatable Read durumlarına izin verirken, Dirty-Read durumuna izin vermez. Şimdi örnek uygulamamızdan iki adet çalıştırılabilir ve aşağıdaki tabloda yer alan hareketleri sırasıyla uygulayalım. Elbette tüm örneklerimizde ilk olarak Başlat başlıklı butona basarak Transaction' ların başlamasını sağlamalıyız.

IsolationLevel.ReadCommitted		
	Transaction 1	Transaction 2
Phantoms	Veri Çeker. (Bak başlıklı button)	
		Veri Çeker. (Bak başlıklı button)
		Yeni satır ekler. (Ekle başlıklı button)
		Transaction onaylanır. Commit (Onayla başlıklı button)
	Tekrardan Veri Çeker. Phantoms durumu oluşur.	

Phantoms

Baslat Bak

☐ ReadUncommitted ☐ Serializable ☐ ReadCommitted ☐ RepeatableRead

Onayla GeriAl

ReadCommitted

Isim Soyisim ID

Ekle Güncelle Bak

ID	ISIM	SOYISIM
59	SEFER	ALGAN
73	Giris	1
76	DENEME	DENEME
78	Burak	Selim
79	Okan	Barlas
80	Feryat	Olcay
81	Resul	Çavuşoğlu

Transaction 1

Phantom Satır

Phantoms

Baslat Bak

☐ ReadUncommitted ☐ Serializable ☒ ReadCommitted ☐ RepeatableRead

Onayla GeriAl

ReadCommitted

Isim Soyisim ID

Resul Çavuşoğlu Ekle Güncelle Bak

ID	ISIM	SOYISIM
70	bos bos	bos
58	Ahmet	Nacaroglu
59	SEFER	ALGAN
73	Giris	1
76	DENEME	DENEME
78	Burak	Selim
79	Okan	Barlas
80	Feryat	Olcay

Transaction 2

Şimdide, Non-Repeatable Read durumunu inceleyelim. Bunun içinde, yine aynı uygulamadan iki tane başlatabilir yada halen çalışan uygulamalarda açık kalan Transaction' ları onaylayarak yeni baştan oluşturabilirsiniz. Bu kez aşağıdaki tabloda izleyen adımları gerçekleştireceğiz.

IsolationLevel.ReadCommitted		
Non-	Transaction 1	Transaction 2

Repeatable Read	Belirli bir satır veri çekilir. (Örneğin ID=58)	
		Aynı satır çekilir. (ID=58)
		Bu satırdaki verilerde değişiklikler yapılır.
		Transaction onaylanır. Commit.
	Aynı satıra tekrar bakılır. Non-Repeatable Read durumu oluşur.	

Phantoms

Baslat

Bak

☐ ReadUncommitted
 ☒ ReadCommitted
 ☐ Serializable
 ☐ RepeatableRead

Onayla

GeriAl

ReadCommitted

Isim

Soyisim

ID

73

Ekle

Güncelle

Bak

ISIM	SOYISIM
Corc	Bus

Transaction 1

Phantoms

Baslat

Bak

☐ ReadUncommitted
 ☒ ReadCommitted
 ☐ Serializable
 ☐ RepeatableRead

Onayla

GeriAl

ReadCommitted

Isim

Soyisim

ID

Corc

Bus

73

Ekle

Güncelle

Bak

ISIM	SOYISIM
Resul	Çavusoglu

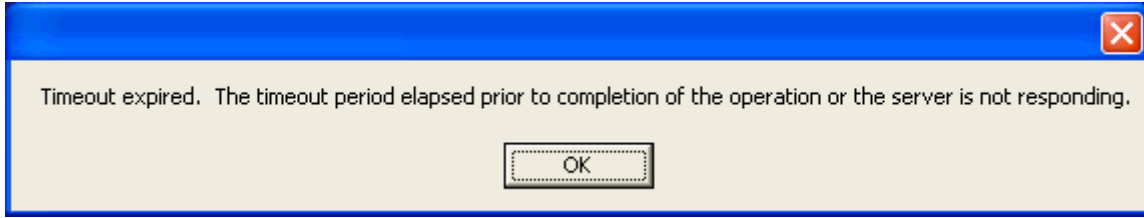
Transaction 2

ReadCommitted seviyesi ile ilgili olarak son durum ise Dirty Read durumudur. Bu izolasyon seviyesi, Dirty Read durumlarının oluşmasına izin vermez. Hatırlayacağınız gibi bu durumda, eş zamanlı olarak çalışan Transaction' lar da herhangi bir Commit işlemi olmadan söz konusu olan problemler yer almaktadır. Şimdi bu durumu incelemek için aşağıdaki tabloda yer alan işlemleri gerçekleştirelim.

IsolationLevel.ReadCommitted

	Transaction 1	Transaction 2
Dirty Read	Belirli bir satır veri çekilir. (Örneğin ID=73)	
		Aynı satır çekilir. (ID=73)
		Bu satırdaki verilerde değişiklikler yapılır. Ancak Transaction Commit edilmez.
	Aynı satıra tekrar bakılmak istenir. Hata Oluşur.	

Görüldüğü gibi, aşağıdaki hata bildirisi çalışma zamanında oluşur.



Dolayısıyla şunu söyleyebiliriz. ReadCommitted değerine sahip Transaction' lardan herhangi biri içerisinde yapılan güncelleme, silme veya ekleme gibi işlemlerin, diğer Transaction tarafından görülebilmesi için, bu değişiklikleri içeren Transaction' ın Commit edilmesi veya RollBack edilmesi gerekmektedir. Bu nedenle, ReadCommitted değeri, Dirty Read durumuna izin vermez.

Gelelim, ReadUncommitted değerine. Bu değer ReadCommitted değerinden tek farkı Dirty Read durumuna izin vermesidir. Bu durumu analiz etmek için, aşağıdaki tablodaki işlemleri sırasıyla uygulayalım. (Transaction' ları Başlat başlıklı Button ile başlatmadan önce, ReadUncommitted RadioButton kontrolünün seçili olmasına dikkat edelim.)

IsolationLevel.ReadUncommitted		
	Transaction 1	Transaction 2
Dirty Read	Belirli bir satır veri çekilir. (Örneğin ID=70)	
		Aynı satır çekilir. (ID=70)
		Bu satırdaki verilerde değişiklikler yapılır. Ancak Transaction Commit edilmez.
	Aynı satıra tekrar bakılmak istenir. (ID=70) Meydana gelen yeni değişiklikler görülür.	
		Yapılan işlemler geri alınır. RollBack.
	Bu Transaction' da gerçekleşmemiş değişiklikler görünmeye devam eder. Dirty Read durumu.	

Dolayısıyla ReadUncommitted değerinin en önemli özelliği, Dirty Read durumuna neden olacak işlemlere izin vermesidir. Başka bir deyişle, çalışan iki Transaction' dan herhangi birinde yapılan değişikliklerin diğer Transaction tarafından görülmesi için, işlemlerin mutlaka Commit edilmesi veya RollBack edilmesi gerekmez.

Sırada, RepeatableRead değeri var. Bu değeride ReadCommitted ile kıyaslayarak anlamaya çalışmak daha mantıklıdır.

RepeatableRead değeri, sadece phantoms durumlarına izin verir. Diğer durumların oluşması halinde, yine zaman aşımı nedeniyle

bir `SqlException` istisnası fırlatılır. `IsolationLevel`' ların belirlenmesi ile ilgili önemli olan bir diğer değerde `Serializable` değeridir. Bu değer, `Phantoms`, `Non-Repeatable Read` ve `Dirty Read` durumlarından herhangi birisinin oluşmasına izin vermez. Örneğin aşağıdaki tabloda yer alan işlemleri yapmaya çalıştığımızı düşünelim.

IsolationLevel.Serializable		
	Transaction 1	Transaction 2
Phantoms	Tablodan veriler çekilir.	
		Tablodan veriler çekilir.
		Yeni bir satır eklenmeye çalışılır. Ancak buna izin verilmez. Çünkü satırlar, <code>Serializable</code> değeri nedeniyle diğer Transaction tarafından kilitlenmiştir.

Dolayısıyla `Serializable` değerinin eş zamanlı Transaction' lar için sadece veri bakmaya izin verdiğini söyleyebiliriz. Başka bir deyişle, aynı anda çalışan iki Transaction' dan herhangi biri, Transaction' lardan diğeri açık olduğu sürece veri girişi, düzenlemesi veya silme işlemlerini yapamaz. Böylece `Sql Server` üzerinde çalışan eş zamanlı Transaction' lar için var olabilecek sorunları değerlendiren izolasyon seviyelerini kısaca görmüş olduk. İlerleyen makalelerimizde, `Sql Server` kilitlerinin `Ado.Net` içindeki yerini incelemeye çalışacağız. Hepinize mutlu günler dilerim.

NET Remoting' i Kavramak - 2

Değerli Okurlarım, Merhabalar.

Bu makalemizde, daha önceden değinmiş olduğumuz `.net remoting` ile ilgili olarak çok basit bir örnek geliştirmeye çalışacağız. `Remoting`' de amaç, istemcilerin uzak nesnelere ait üyelere erişebilmelerini ve kullanabilmelerini sağlamaktır. Dolayısıyla, `remoting` sistemi söz konusu olduğunda, `remote object`, `server channels` ve `client channels` kavramları önem kazanır. Olaya bu açıdan baktığımızda ilk olarak bir `remote object` (uzak nesne) geliştirmemiz gerektiği ortadadır. Daha sonra, bu nesneyi kullanmak isteyen istemcileri dinleyecek bir server programını yazmamız gerekir. Bu server programı aslında, `remote object`' i barındıran (host) bir hizmet programı olacaktır. İstemcilerin tek yapması gereken uzak nesneye ait bir örneği, çalıştıkları sistemde kullanarak bir proxy nesnesi oluşturmak ve bu nesne üzerinden server programa istekte bulunarak ilgili uzak nesneye ait metodları çalıştırmaktır. İşte bu noktada server ve client uygulamalardaki channel nesneleri önem kazanır.

Biz bugünkü örneğimizde, `TCP` protokolü üzerinden haberleşen bir `remoting` uygulaması geliştirmeye çalışacağız. Öncelikle işe, `remote object` (uzak nesne) sınıfını tasarlayarak başlayalım. Konuyu daha net anlayabilmek için, uygulamalarımızı editoründe geliştireceğiz. Şimdi aşağıdaki kodlardan oluşan, `UzakNesne.cs` kod dosyasını oluşturalım.

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

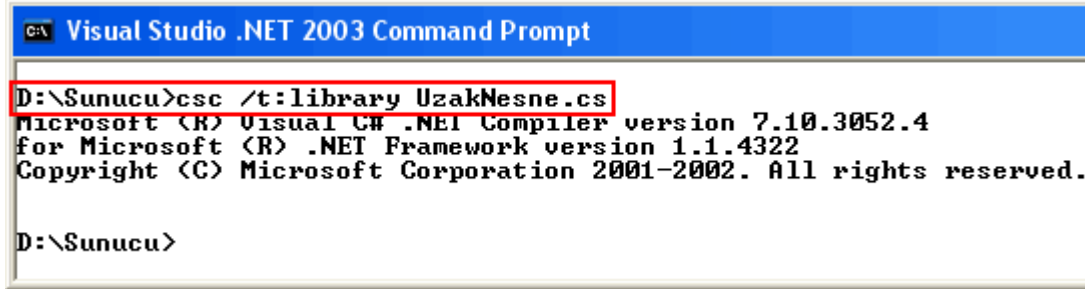
namespace UzakNesne
{
    public class Musteriler:System.MarshalByRefObject
```

```

{
    public string MusteriBul(string CustID)
    {
        SqlConnection con=new SqlConnection("data source=localhost;initial
        catalog=Northwind;integrated security=SSPI");
        SqlCommand cmd=new SqlCommand("SELECT * FROM Customers WHERE
        CustomerID='"+CustID+"' ",con);
        con.Open();
        SqlDataReader dr=cmd.ExecuteReader();
        dr.Read();
        string Bulunan=dr["CompanyName"]+" "+dr["ContactName"]+" "+dr["Phone"];
        con.Close();
        return Bulunan;
    }
}

```

Burada görüldüğü gibi remote object (uzak nesne) sınıfı için en önemli unsur, bu sınıfın System.MarshalByRefObject sınıfından türetilmiş olmasıdır. Normal şartlarda geliştirdiğimiz bu nesneye, bulunduğu sistemin application domain' i içerisinde direkt olarak erişebiliriz. Ancak, bu nesnenin application domain' in dışındaki bir domainden kullanılabilmesini sağlamak yani remoting desteğini vermek için MarshalByRefObject sınıfından türetmemiz gerekmektedir. Bu sayede istemci uygulama, bu nesneye ait proxy nesnesi üzerinden mesaj gönderebilecek ve alabilecektir. Şimdi geliştirdiğimiz bu assembly' ı bir dll olarak aşağıdaki gibi derleyeylim.



```

C:\ Visual Studio .NET 2003 Command Prompt
D:\Sunucu>csc /t:library UzakNesne.cs
Microsoft (R) Visual C# .NET Compiler version 7.10.3052.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

D:\Sunucu>

```

Artık uzak nesnemizde sahip olduğumuza göre, bu nesneyi kullanacak istemcileri takip edecek, bir başka deyişle dinleyecek (listening) bir server (sunucu) uygulaması yazabiliriz. Bir sunucu uygulaması için en önemli unsurlar, dinleme işlemi için hangi kanalın, hangi ayarlar ile kullanılacağı ve remote object (uzak nesne) sınıfına ait bilgilerin ne olduğundan oluşmaktadır. Server (sunucu) için gerekli bu bilgileri uygulama içerisinde programatik olarak belirleyeceğimiz gibi, xml tabanlı bir konfigürasyon dosyası yardımıyla tanımlayabiliriz. XML tabanlı konfigürasyon dosyasının, programatik olan tekniğe göre en büyük avantajı, kanal ve remote object (uzak nesne) ile ilgili düzenlemelerde, kodun yeniden derlenmesine gerek kalınmamasıdır. Bir konfigürasyon dosyası, config uzantılıdır. Bu makalemizde kullanacağımız Server (sunucu) uygulamaya ait konfigürasyon dosyası Sunucu.config isminde olup, aşağıdaki satırlardan oluşmaktadır.

<configuration>

```

<system.runtime.remoting>
  <application name="MusteriUygulama">
    <service>
      <wellknown mode="SingleCall" type="UzakNesne.Musteriler,UzakNesne" objectUri="UzakNesne"/>
    </service>
    <channels>
      <channel ref="tcp server" port="1000"/>
    </channels>
  </application>
</system.runtime.remoting>
</configuration>

```

Şimdi bu xml dosyasının içeriğini kısaca incelemeye çalışalım. Öncelikle ana eleman <configuration> dir. Burada server (sunucu) uygulamasına ait remote object (uzak nesne) ve channel (kanal) ayarlamaları mutlaka, <system.runtime.remoting> elemanı içinde yapılmalıdır. <application> elemanı içinde name özelliği ile server (sunucu) uygulamamızın adını belirtiyoruz.

Asıl önemli olan iki eleman <application> elemanı altında yer alan <service> ve <channels> elemanlarıdır. <service> elemanı altında <wellknown elemanı ile, remote object' e ait bilgiler belirtilir. mode niteliğinde, SingleCall ile, sunucu tarafı etkinleştirme tekniği belirtilmektedir. Bu anlamda SingleCall ile, istemci tarafından yapılan her metod çağırısına karşılık birer sunucu nesnesi örneği oluşturulması sağlanmaktadır. Diğer alternatif ise, Singleton tekniğidir ki bu tekniğe göre sunucu nesneyi kullanan kaç istemci olursa olsun, her bir istemcinin metod çağırımları sunucu tarafındaki tek bir nesne örneği tarafından yönetilmektedir.

type niteliğinde remote object (uzak nesne) için, bu tipin hangi isim alanı ve sınıfa ait olduğu belirtilir. type niteliğindeki ikinci parametre ise, uzak nesnenin bulunduğu assembly' ın adıdır. objectUri ile, istemcilerin uzak nesne için kullanacakları endpoint (son nokta) ismi belirtilmektedir. <channel elemanında, hangi protokol üzerinden ve hangi port' tan dinleme yapılacağı belirtilir. Burada sistemdeki machine.config dosyasında daha önceden tanımlanmış olan tcp server protokolünün, 1000 numaralı portu kullanacağı belirtilmiştir. Machine.config dosyasını D:\WINDOWS\Microsoft.NET\Framework\vx.x.xxx\CONFIG klasöründe bulabilirsiniz. Bu dosyada <channels> elemanı altında önceden tanımlanmış remoting protokolleri yer almaktadır.

```

<channels>
  <channel id="http" type="System.Runtime.Remoting.Channels.Http.HttpChannel, System.Runtime.Remoting,
Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <channel id="http client" type="System.Runtime.Remoting.Channels.Http.HttpClientChannel, System.Runtime.Remoting,
Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <channel id="http server" type="System.Runtime.Remoting.Channels.Http.HttpServerChannel, System.Runtime.Remoting,
Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <channel id="tcp" type="System.Runtime.Remoting.Channels.Tcp.TcpChannel, System.Runtime.Remoting, Version=1.0.5000.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <channel id="tcp client" type="System.Runtime.Remoting.Channels.Tcp.TcpClientChannel, System.Runtime.Remoting,
Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <channel id="tcp server" type="System.Runtime.Remoting.Channels.Tcp.TcpServerChannel, System.Runtime.Remoting,

```

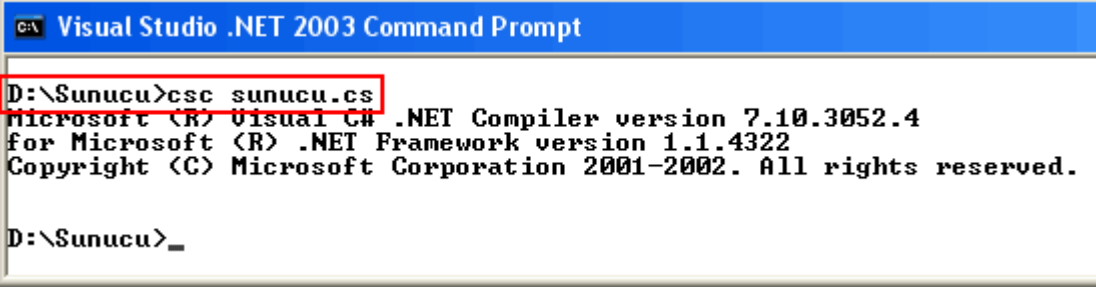
```
Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
</channels>
```

Burada Machine.config içindeki tcp server protokolüne ait tanımlamada, System.Runtime.Remoting.Channels.Tcp.TcpServerChannel tipinde belirtildiğine dikkat edelim. Konfigurasyon dosyasının hazırlanması ile birlikte, artık sunucu uygulamamızı yazabiliriz. Sunucu uygulama, bu konfigurasyon dosyasını kullanarak, belirtilen protokol ve port üzerinden istemci taleplerini (mesajlarını) dinlemeye alacak ve gelen bu talepleri, yine konfigurasyon dosyasında belirtilen uzak nesneye ait örneğe yönlendirecektir. Nesneden dönen cevaplarda, yine bu konfigurasyon dosyasında belirtilen protokol ve port ile istemcilere gönderilecektir. Gelelim server(sunucu) uygulamamızın kodlarına.

```
using System;
using System.Runtime.Remoting;

namespace Sunucu
{
    public class SunucuDinler
    {
        public static void Main(string[] args)
        {
            RemotingConfiguration.Configure("Sunucu.config");
            Console.WriteLine("Dinlemeye baslandı...Çıkmak için bir tusa basın");
            Console.ReadLine();
        }
    }
}
```

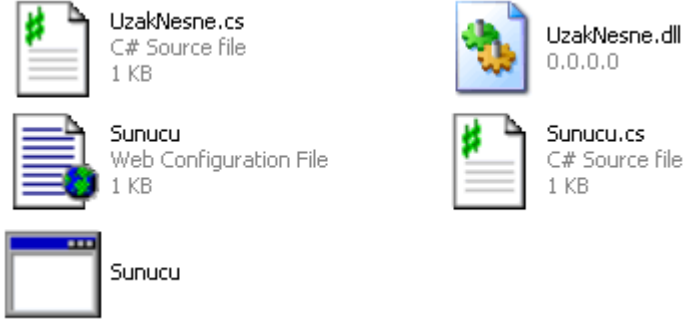
Burada görüldüğü gibi, sunucu uygulamamız konfigurasyon dosyasındaki bilgilere RemotingConfiguration sınıfının Configure metodu ile ulaşmaktadır. İzleyen satırlardaki WriteLine ve ReadLine metodları ile, uygulamanın biz son verene kadar 1000 nolu Tcp Server portunu dinlemesi sağlanmıştır. Şimdi yazdığımız bu uygulamayı exe olarak derleyelim.



```
C:\ Visual Studio .NET 2003 Command Prompt
D:\Sunucu>csc sunucu.cs
Microsoft (R) Visual C# .NET Compiler version 7.10.3052.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

D:\Sunucu>_
```

Bu işlemlerin ardından, sunucu nesnemizin, sunucu uygulamamızın ve konfigurasyon dosyamızın aynı klasörde olmalarına dikkat edelim.



Artık sunucu tarafındaki işlemlerimizi bitirmiş olduk. Şimdi sunucu tarafındaki nesneyi kullanacak istemci uygulamayı tasarlayalım. Elbetteki, istemci uygulamada, mesajlarını Tcp protokolünü baz alarak bir kanal üzerinden göndermek zorundadır. Çünkü sunucu uygulama Tcp protokolünü baz alarak dinleme gerçekleştirecektir. Bunu sağlamak için öncelikle, istemci tarafında, client channel (istemci kanal) nesnesine ihtiyacımız vardır. İlave olarak, kullanacağımız uzak nesnenin bir koypasını, istemci uygulamanın assembly'ının bulunduğu klasöre koymamız gerekiyor. Bir istemci uygulama için önemli olan unsurlar, kanal nesnesi ayarları ve uzak nesneye ait ayarlardır. Bu ayarlamaları programatik olarak yapabileceğimiz gibi bir konfigürasyon dosyasında da yapabiliriz. Bu amaçla öncelikle aşağıdaki istemci.config dosyasını oluşturalım.

```
<configuration>
  <system.runtime.remoting>
    <application name="Istemci">
      <client url="tcp://localhost:1000/Sunucu">
        <wellknown type="UzakNesne.Musteriler,Musteriler" url="tcp://localhost:1000/Sunucu/Musteriler"/>
      </client>
      <channels>
        <channel ref="tcp client"/>
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

Burada <client elemanında url niteliğinde, sunucu uygulamanın tcp üzerinden erişilebilen adresi verilmiştir. Böylece istemci mesajlarını dinleyecek sunucu uygulamasının sunucu,protokol,port ve assembly adı bilgileri belirlenmektedir. <wellknown elemanında proxy nesnesi ile iletişim kuracak uzak nesneye ait bilgiler yer almaktadır. Özel olarak, uzak nesnenin tam olarak adresi url bilgisinde belirtilmiştir. Bu url bilgisi, istemci uygulamadaki proxy nesnesinin mesajlaşmak amacıyla kullanacağı karşı nesne adresini tam olarak belirtmektedir. <channels> elemanında ise, client channel nesnesi belirtilmiştir. Bu nesne yine ref niteliği ile, sistemde machine.config dosyasında daha önceden tanımlanmış tcp client niteliğini referans etmektedir. Artık istemci uygulamamızda yazabiliriz.

```
using System;
using System.Runtime.Remoting;
```

```

namespace istemciUygulama
{
    public class istemci
    {
        public static void Main(string[] args)
        {
            RemotingConfiguration.Configure("istemci.config");
            UzakNesne.Musteriler m=new UzakNesne.Musteriler();
            string sonuc=m.MusteriBul("ALFKI");
            Console.WriteLine(sonuc);
            Console.ReadLine();
        }
    }
}

```

Görüldüğü gibi istemci uygulamada, konfigürasyon ayarlarını almak için RemotingConfiguration sınıfının Configure metodunu kullanır. Bundan sonra tek yapılan uzak nesne örneğini oluşturmak ve ilgili metodu çağırmasıdır. Yazdığımız bu istemci.cs dosyasını aşağıdaki şekilde derleyeylim.

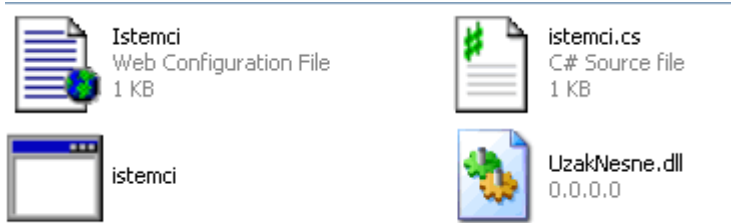
```

C:\ Visual Studio .NET 2003 Command Prompt
D:\RemoteIstemci>csc /reference:uzaknesne.dll istemci.cs
Microsoft (R) Visual C# .NET Compiler version 7.10.3052.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

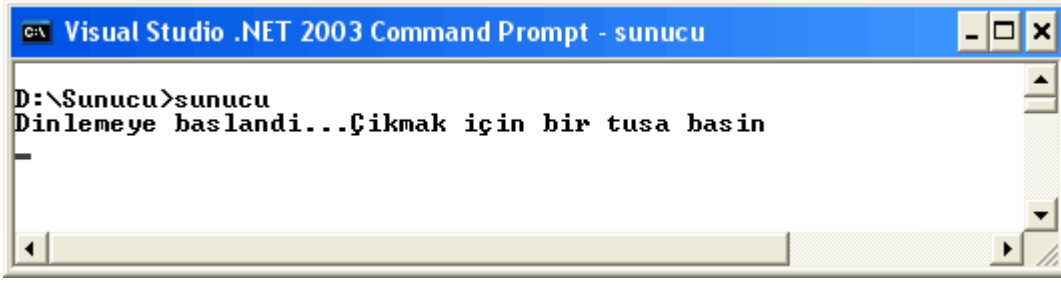
D:\RemoteIstemci>_

```

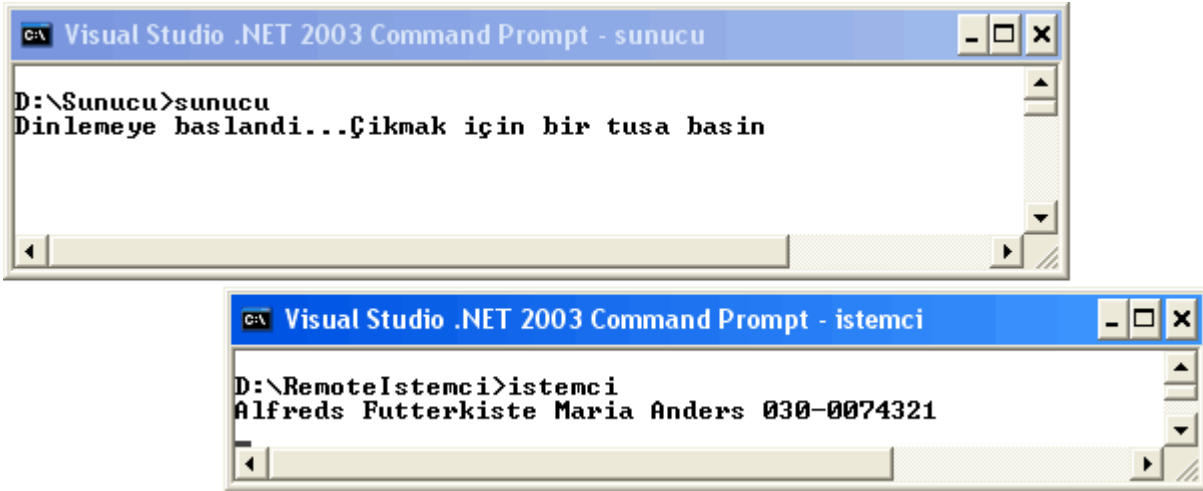
Burada dikkat edilecek olursa, uzaknesne.dll dosyası assembly'a referans olarak bildirilmiştir. Dolayısıyla, uzaknesne.dll' inin bir kopyasının istemci uygulamanın bulunduğu klasörde olması gerektiğine dikkat etmeliyiz.



Şimdi remoting sistemimizi bir test edelim. Öncelikle sunucu uygulamamızı, istemcilerden gelecek talepleri dinlemek amacıyla başlatmamız gerekiyor.



Bu işlemin ardından istemci uygulamamızda çalıştıralım. Sonuçlar aşağıdaki gibi olacak ve sql sunucusunda yer alan Northwind veritabanındaki Customers tablosundaki CustomerID değeri ALFKI alan satır bilgileri elde edilecektir.



Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Transaction' larda DeadLock Kavramı

Değerli Okurlarım, Merhabalar.

Bu makalemizde, eş zamanlı olarak çalışan Transaction' larda meydana gelebilecek DeadLock durumunu incelemeye çalışacağız. Öncelikle DeadLock teriminin ne olduğunu anlamaya çalışalım. DeadLock, aynı zamanlı çalışan Transaction' ların, belirli satır(ları) kilitlemeleri sonucunda ortaya çıkabilecek bir durumdur. DeadLock terimini kavrayabilmenin en iyi yolu aşağıdaki gibi gelişebilecek bir senaryoyu zihnimizde canlandırmakla mümkündür. Bu senaryoda söz konusu olan iki tablomuz mevcuttur. Bu tablolar Sql sunucusunda Northwind veritabanı altında oluşturulmuş olup Field(alan) yapıları aşağıdaki gibidir.

Start Page	dbo.Musteriler...RKI.Northwind)		dbo.Pers
	Column Name	Data Type	Length
	MusteriID	int	4
	MusteriMail	nvarchar	50
	MusteriTel	nvarchar	50

Şekil 1. Musteriler Tablosu.

Start Page	dbo.Personel : ...URKI.Northwind)	dbo.Must
Column Name	Data Type	Length
PersonelID	int	4
PersonelAd	nvarchar	50
PersonelSoyad	nvarchar	50

Şekil 2. Personel Tablosu.

Şimdi senaryomuzu tasarlayarak DeadLock kavramını anlamaya çalışalım. Her iki tabloyu ayrı ayrı kullanan ve eş zamanlı olarak çalışan iki Transaction' ımız olduğunu düşünelim. Burada önemli olan bu iki Transaction' ın aynı anda çalışıyor olmalarıdır.

DeadLock Senaryosu	
1nci Adım	Transaction 1 başlatılır.
2nci Adım	Transaction 2 başlatılır.
3üncü Adım	Transaction 1 Personel tablosunda PersonelID değeri 78 olan satırı kitler ve günceller.
4üncü Adım	Transaction 2 Musterilre tablosunda MusteriID değeri 1000 olan satırı kitler ve günceller.
5inci Adım	Transaction 1 Musteriler tablosundaki MusteriID değeri 1000 olan satırı güncellemek ister. Ancak, Transaction 2 bu satırı kitlediğinden, varsayılan Lock TimeOut süresi kadar bu kilidin açılmasını bekler. Bu süre sonuna kadar Transaction 2' nin işlemlerini onaylaması veya geri alması beklenir.
6ncı Adım	Transaction 2 Personel tablosundaki PersonelID değeri 78 olan satırı güncellemek ister. Ancak bu durumda, Transaction 1 bu satırı kitlediğinden yine varsayılan Lock TimeOut süresi kadar bu kilidin açılmasını bekler. Bu süre sonuna kadar Transaction 1' in işlemlerini onaylaması veya geri alması beklenir.

Bizim için önemli olan adımlar, 5inci ve 6ncı adımlardır. Nitekim bu adımlar eş zamanlı olarak gerçekleştirildiklerinden, iki Transaction da birbirinin kilitlerinin açılmasını beklemek durumundadır. İşte bu noktada DeadLock oluşur. Nitekim süreler sona erinceye kadar herhangi bir Transaction sahip olduğu işlemleri ne onaylamış (Commit) nede geri almıştır (RollBack). Dolayısıyla Transaction' lardan birisi, varsayılan olarak Sql sunucusuna göre en maliyetli olanı otomatik olarak RollBack edilecektir. Bu durumda bu Transaction' a ait kilitler ortadan kalktığından kalan Transaction' a ait güncelleme işlemi tamamlanır. Ancak .net ile yazılan uygulamalarda DeadLock oluştuğunda, Transaction' lardan birisi geri alınmakla kalmaz aynı zamanda ortama bir istisna fırlatılır. Dolayısıyla DeadLock durumunda bu istisnada ele alınması gerekirken, DeadLock sonucu çalışmaya devam eden Transaction işlemleri onaylanabilsin.

DeadLock oluşması durumunda, birbirlerini bekleyen Transaction' larda, bekleme sürelerini ayarlayabilir ve hangi Transaction' ın daha önce RollBack edilmesi gerektiğine karar verebiliriz. Bunun için, T-Sql' in LOCK_TIMEOUT ve DEADLOCK_PRIORITY anahtar sözcükleri kullanılır. Bir Transaction' ın başka bir Transaction' da oluşan kilidi ne kadar süre ile beklemesi gerektiğini belirtmek için aşağıdaki gibi bir sql cümlecici kullanılır.

```
SET LOCK_TIMEOUT 3000
```

Burada LOCK_TIMEOUT değeri 3 saniye (3000 milisaniye) olarak belirtilmiştir. Diğer yandan, bir Transaction için DeadLock önceliğini aşağıdaki gibi bir sql cümlecisi ile belirtebiliriz.

```
SET DEADLOCK_PRIORITY LOW
```

Bu sql cümlecisini kullanan komutun çalıştığı Transaction, DeadLock oluşması durumunda, ilk olarak RollBack edilecek Transaction olacaktır. DEADLOCK_PRIORITY anahtar sözcüğünün alabileceği diğer değerler NORMAL dir. Bu durumda, Transaction' lardan en maliyetli olanı RollBack edilir. DEADLOCK_PRIORITY için varsayılan değer NORMAL olarak belirlenmiştir. Şimdi dilerseniz DeadLock oluşmasını daha iyi izleyebileceğimiz bir örnek geliştirmeye çalışalım. Bu durumu simule edebilmek için, aynı anda çalışan Transaction iş parçalarına ihtiyacımız olacak. Yani DeadLock senaryosunda belirtmiş olduğumuz güncelleme işlemlerinin aynı zamanda çalışıyor olması gerekli. Bunu sağlayabilmek için bu güncelleme işlemlerini birer Thread içinde çalıştıracacağız. Uygulamamız basit bir Console Application olacak. Şimdi aşağıdaki uygulama kodlarını oluşturalım.

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

namespace DeadLockTest
{
    class Deadlock
    {
        /* İki ayrı Transaction için iki ayrı SqlConnection nesnesi oluşturuyoruz. Bununla birlikte, iki Transaction içindeki komutları icra edecek iki adet SqlCommand nesnesi ve Transactionlar içinde iki adet SqlTransaction nesnesi tanımlıyoruz.*/
        public static SqlConnection conT1 =new SqlConnection("server=localhost;database=Northwind;integrated security=SSPI");
        public static SqlConnection conT2 =new SqlConnection("server=localhost;database=Northwind;uid=sa;integrated security=SSPI");
        public static SqlCommand cmdT1;
        public static SqlCommand cmdT2;
        public static SqlTransaction T1;
        public static SqlTransaction T2;

        /* Bu metod, Personel tablosunda güncelleme işlemini yapıyor. Komut, conT1 SqlConnection' i üzerinden, T1 isimli SqlTransaction' da çalışacak şekilde oluşturuluyor. Sonra bu komut yürütülüyor. Metod deadlock senaryomuzun 3ncü adimini gerçekleştiriyor.*/
        public static void GuncellePersonelT1()
        {
            Console.WriteLine("PersonelID=78 olan satirdaki PersonelAd alaninin degeri DENEME yapiliyor...");
            cmdT1=new SqlCommand("UPDATE Personel SET PersonelAd = 'DENEME' WHERE PersonelID = 78", conT1,T1);
            int sonuc = cmdT1.ExecuteNonQuery();
        }
    }
}
```

```

        Console.WriteLine(sonuc+" guncellendi. PersonelAd = DENEME yapildi.");
    }

    /* Bu metod ile DeadLock senaryomuzun 4ncü adimi gerçekleştiriliyor.*/
    public static void GuncelleMusterilerT2()
    {
        Console.WriteLine("MusteriID=1000 olan satirdaki MusteriMail alaninin degeri DENEME@DENEME.COM yapiliyor...");
        cmdT2=new SqlCommand("UPDATE Musteriler SET MusteriMail = 'DENEME@DENEME.COM' WHERE MusteriID =
1000",conT2,T2);
        int sonuc = cmdT2.ExecuteNonQuery();
        Console.WriteLine(sonuc+" guncellendi. MusteriMail = DENEME@DENEME.COM yapildi.");
    }

    /* Bu metod ile DeadLock senaryomuzun 5nci adimi gerçekleştiriliyor.*/
    public static void GuncelleMusterilerT1()
    {
        Console.WriteLine("MusteriID=1000 olan satirdaki MusteriMail alaninin degeri MAIL@MAIL.COM yapiliyor...");
        cmdT1 =new SqlCommand("UPDATE Musteriler SET MusteriMail = 'MAIL@MAIL.COM' WHERE MusteriID =
1000",conT1,T1);
        int sonuc = cmdT1.ExecuteNonQuery();
        Console.WriteLine(sonuc+" guncellendi. MusteriMail = MAIL@MAIL.COM yapildi.");
    }

    /* Bu metod ilede DeadLock senaryomuzun 6nci adimi gerçekleştiriliyor.*/
    public static void GuncellePersonelT2()
    {
        Console.WriteLine("PersonelID=78 olan satirdaki PersonelAd alaninin degeri ISIM yapiliyor...");
        cmdT2=new SqlCommand("UPDATE Personel SET PersonelAd = 'ISIM' WHERE PersonelID = 78", conT2,T2);
        int sonuc = cmdT2.ExecuteNonQuery();
        Console.WriteLine(sonuc+" guncellendi. PersonelAd = ISIM yapildi.");
    }

    public static void Main()
    {
        /* Baglantimiz açılıyor ve ilk transaction başlatılıyor. Ardından bu Transaction için LOCK_TIMEOUT degeri belirlenerek,
        kilitlerin ne kadar süre ile bekleneceğini belirten sql komutu çalıştırılıyor. Süre olarak 3 saniye veriliyor.*/
        conT1.Open();
        T1 = conT1.BeginTransaction();
        cmdT1 = conT1.CreateCommand();

```

```
cmdT1.Transaction = T1;
cmdT1.CommandText = "SET LOCK_TIMEOUT 3000";
cmdT1.ExecuteNonQuery();
```

/* İkinci transaction için gerekli bağlantı açılıyor, transaction başlatılıyor ve LOCK_TIMEOUT süresi 3 saniye olarak belirleniyor.*/

```
conT2.Open();
T2 = conT2.BeginTransaction();
cmdT2 = conT2.CreateCommand();
cmdT2.Transaction = T2;
cmdT2.CommandText = "SET LOCK_TIMEOUT 3000";
cmdT2.ExecuteNonQuery();
```

/* İzleyen sql cümlecigi ile, DeadLock_Priority değeri LOW olarak belirleniyor. Yani, bir deadLock oluşması durumunda, cmdT2 nin içinde çalıştığı Transaction RollBack edilecektir.*/

```
cmdT2.CommandText = "SET DEADLOCK_PRIORITY LOW";
cmdT2.ExecuteNonQuery();
```

/*DeadLock senaryomuzdaki update işlemlerini gerçekleştirecek olan metodlarımız için Thread nesneleri oluşturuluyor ve daha sonra bu Thread'ler başlatılıyor.*/

```
Thread Thread1 = new Thread(new ThreadStart(GuncellePersonelT1));
Thread Thread2 = new Thread(new ThreadStart(GuncelleMusterilerT2));
Thread Thread3 = new Thread(new ThreadStart(GuncelleMusterilerT1));
Thread Thread4 = new Thread(new ThreadStart(GuncellePersonelT2));
```

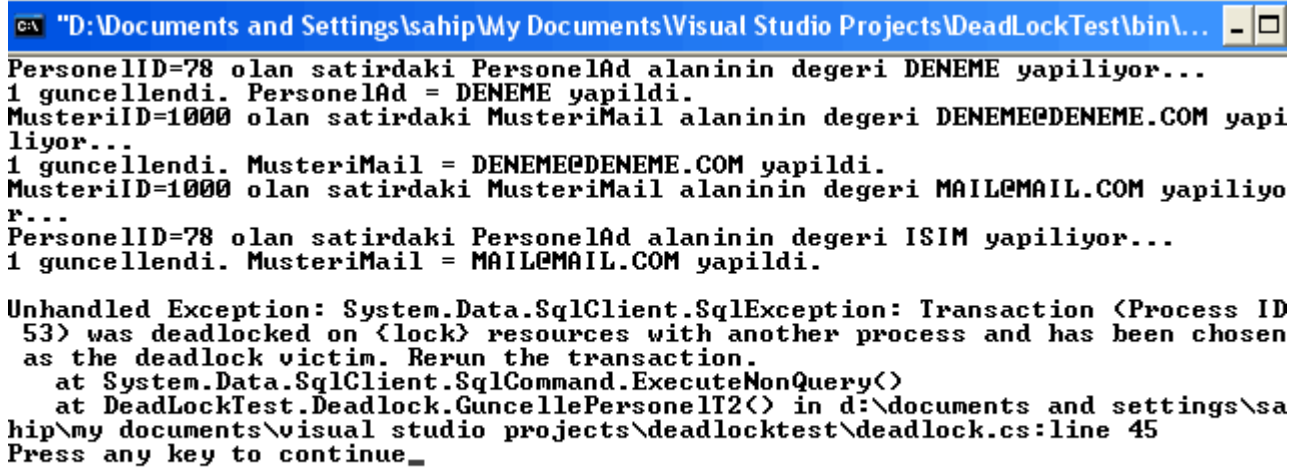
```
Thread1.Start();
Thread2.Start();
Thread3.Start();
Thread4.Start();
```

```
Console.ReadLine();
```

```
}
```

```
}
```

```
}
```



```

C:\> "D:\Documents and Settings\saip\My Documents\Visual Studio Projects\DeadLockTest\bin\...
PersonelID=78 olan satirdaki PersonelAd alaninin degeri DENEME yapiliyor...
1 guncellendi. PersonelAd = DENEME yapildi.
MusteriID=1000 olan satirdaki MusteriMail alaninin degeri DENEME@DENEME.COM yap
liyor...
1 guncellendi. MusteriMail = DENEME@DENEME.COM yapildi.
MusteriID=1000 olan satirdaki MusteriMail alaninin degeri MAIL@MAIL.COM yapiliyo
r...
PersonelID=78 olan satirdaki PersonelAd alaninin degeri ISIM yapiliyor...
1 guncellendi. MusteriMail = MAIL@MAIL.COM yapildi.

Unhandled Exception: System.Data.SqlClient.SqlException: Transaction (Process ID
53) was deadlocked on {lock} resources with another process and has been chosen
as the deadlock victim. Rerun the transaction.
   at System.Data.SqlClient.SqlCommand.ExecuteNonQuery()
   at DeadLockTest.Deadlock.GuncellePersonelT2() in d:\documents and settings\sa
hip\my documents\visual studio projects\deadlocktest\deadlock.cs:line 45
Press any key to continue_

```

Şekil 3. Programın Çalışmasının Sonucu.

Uygulamayı çalıştırdığımızda, ilk Thread ve ikinci Thread' ler içinde yürütülen update işlemlerinin gerçekleştirildiğini görürüz. Daha sonra üçüncü Thread için güncelleme işlemi yapılırken, bu Thread' in çalıştığı Transaction 3 saniye süreyle diğer Transaction' daki kilidin açılmasını bekleyecektir. Süre sonunda kilit halen daha açılmamış olacağından (ki kilidin açılması Commit veya RollBack işlemini gerektirir.) DEADLOCK_PRIORITY değeri LOW olarak belirlenen ikinci Transaction RollBack edilecektir. Bununla birlikte SqlException türünden bir istisna da ortama fırlatılır. Bu istisnada bir DeadLock oluştuğu ve prosesler içinde çalışan Transaction' lardan birisinin kurban edileceği belirtilir. Elbette burada istisnayı kontrol etmediğimiz için her iki Transaction içindeki işlemler RollBack edilecektir. Ana amacımız, DeadLock senaryosunun ne zaman gerçekleşeceği ve neler olacağıdır. Böylece DeadLock durumunun nasıl oluştuğunu ve nelere yol açtığını çok kısa ve basit olarak incelemeye çalıştık. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

NET Remoting' i Kavramak – 3

Değerli Okurlarım, Merhabalar.

Bu makalemizde, uzak nesneler üzerindeki metodlara asenkron olarak nasıl erişebileceğimizi kısaca incelemeye çalışacağız. Remoting ile ilgili bir önceki makalemizde, çok basit haliyle uzak nesnelerin, istemciler tarafından nasıl kullanılabildiğini incelemiştik. Geliştirmiş olduğumuz örnekte, uzak nesne üzerindeki metoda senkron olarak erişmekteydik. Yani, uzak nesnedeki metodun işleyişi bitene kadar, istemci uygulama kısa sürelide olsa duraksıyordu. Ancak bazı zamanlarda, uzak nesneler üzerinde işleyecek olan metodlar, belirli bir süre zarfında gerçekleşebilecek uzunlukta işlemlere sahip olabilirler. Böyle bir durumda istemci uygulamalar, metodların geri dönüş değerlerini beklemek zorunda kalabilirler. Oysaki, uzak nesneye ait metodlar bir yandan çalışırken, diğer yandanda istemci uygulamadaki izleyen kod satırlarının eş zamanlı olarak çalışması istenebilir. Bunu sağlamak için, uzak nesne metodlarına asenkron olarak erişilir.

Uzak nesne metodlarına asenkron olarak erişim, istemci uygulamalar için oldukça kullanışlıdır. Elbette bazı durumlarda senkron erişim tercih edilir. Öyleki, uzak nesneye ait metodun sonucu veya sonuçları, istemci uygulamada izleyen kod satırlarında kullanılıyor olabilir veya uzak nesne metodunun sonucu, istemci uygulama içindeki başka metodlara parametre olarak gönderiliyor

olabilir vb... Elbette böyle bir durumda, uzak nesne üzerindeki metoda asenkron olarak erişmek çok mantıklı değildir. Nitekim, uzak nesne metodunun sonucunun veya sonuçlarının etkilediği başka işlemler söz konusudur.

Bu kısa açıklamalardan sonra, asenkron erişim için gerekli olan temel unsurlardanda kısaca bahsedelim. Uzak nesne üzerindeki metodların asenkron olarak çağırılması, normal bir uygulamadaki metodların asenkron olarak çağırılmasından çok da farklı değildir. Örneğin aşağıdaki console uygulamasını ele alalım. Bu uygulamada basit olarak Hesapla isimli metoda asenkron erişim gerçekleştirilmiştir.

```
using System;
using System.Threading;

namespace istemciUygulama
{
    public class Sinif
    {
        public double Hesapla(double a,double b)
        {
            Thread.Sleep(3500);
            return(a*a+b*b);
        }
    }

    public class istemci
    {
        private delegate double Temsilci(double d1,double d2);

        public static void Main(string[] args)
        {
            Sinif nesne=new Sinif();
            Temsilci t=new Temsilci(nesne.Hesapla);
            IAsyncResult res=t.BeginInvoke(4,5,null,null);
            Console.WriteLine("Uygulama çalışıyor...");
            res.AsyncWaitHandle.WaitOne();
            if(res.IsCompleted)
            {
                double sonuc=t.EndInvoke(res);
                Console.WriteLine(sonuc);
            }
            Console.ReadLine();
        }
    }
}
```

```

    }
}

```

Bu uygulamayı derleyip çalıştırdığımızda acaba tam olarak neler olmaktadır? Hesapla isimli metodumuz double tipinden iki parametre alan ve yine double tipinden değer döndüren bir yapıya sahiptir. Bu metod içerisinde, Thread sınıfının sleep metodu kullanılmış ve uygulama yaklaşık olarak 3.5 saniye süre ile duraksatılmıştır. Burada amaç uzun süren bir metod işleyişi gerçekleştirmektir. Metoda asenkron erişimin sağlanabilmesi için, bir delegate nesnesi kullanılmaktadır. Öncelikle delegate nesnemiz tanımlanır.

```
private delegate double Temsilci(double d1,double d2);
```

Delegate nesnesinin metod imzasının, Hesapla metodu ile aynı olduğuna ve tipinde, Hesapla metodunun geri dönüş değeri tipi ile aynı olduğuna dikkat edelim. Gelelim Main metodu içerisindeki kodlara. Öncelikle,

```
Temsilci t=new Temsilci(nesne.Hesapla);
```

satırları ile delegate nesnemiz oluşturulmaktadır. Artık t isimli temsilcimiz, nesne sınıfına ait Hesapla metodunun bellekteki başlangıç adresini temsil etmektedir. İşte bu adımdan sonraki işlemler önemlidir ve asenkron erişim tekniğinin uygulanışını içermektedir.

```
IAsyncResult res=t.BeginInvoke(4,5,null,null);
```

Satırı ile, delegate nesnesi için BeginInvoke metodu çağırılmaktadır. Bu metod görüldüğü gibi 4 parametre almıştır. İlk iki parametre, temsilcinin işaret ettiği metodun kullanacağı iki parametrenin değerini belirtmektedir. Sonraki parametreler ise null olarak bırakılmıştır. Burada olan olay şudur. t isimli temsilcinin işaret ettiği metod 4 ile 5 değerlerini parametre olarak çalışmaya başlamıştır. Lakin, bir metod çağırımından sonra uygulamanın izleyen kod satırlarını devam ettirebilmesi için, metodun işleyişini tamamlamış olması gerekir. Ancak burada, BeginInvoke ile Hesapla metodu çalıştırılmış ve anında ortama IAsyncResult arayüzü türünden bir nesne döndürülmüştür. Nitekim BeginInvoke metodunun geri dönüş değeri IAsyncResult arayüzü tipinden bir nesne örneğidir. Dolayısıyla izleyen satırlardaki kodlar işletilebilecektir. Bunun sonucu olarak ekrana "Uygulama çalışıyor..." ifadesi yazılır. Bu noktadan sonra uygulamada istenilen işlemler yapılabilir.

Tabiki temsilcimizin çalıştırdığı metodun sonucunun bir şekilde alınması gerekir. İlk yapılacak işlem, IAsyncResult arayüzünden nesne örneğinin IsCompleted özelliğinin değerine bakmak olacaktır. Bu değer true ise, asenkron metodun işleyişi tamamlanmış demektir. Bu durumda, asenkron olarak çalışan metodun geri dönüş değerini alabilmek için, t temsilcisinin EndInvoke metodu, uygulama ortamında o anda var olan IAsyncResult arayüzü nesne örneği res parametresi ile çağırılır. Sonuç olarak, asenkron metodun çalışmasının sonucu ürettiği değer elde edilir. Tüm bu işlemler için aşağıdaki kod satırları işletilmiştir.

```
res.AsyncWaitHandle.WaitOne();
if(res.IsCompleted)
{
    double sonuc=t.EndInvoke(res);
    Console.WriteLine(sonuc);
}
Console.ReadLine();
```


Burada ilk satır ile, IAsyncResult arayüzü nesnesinin, çalışan asenkron metodun işleyişini tamamlamasını beklemesi söylenmiştir. Bu kod satırının yazılmasının amacı şudur. Asenkron metodun çalıştırılmaya başlamasından sonra, uygulamada izleyen kod satırları bu örnekte olduğu gibi çoktan tamamlanmış ancak halen daha asenkron metodun işleyişi bitmemiş olabilir. Bu durumda if döngüsü gerçekleşmeyeceği için metodun geri dönüş değeride alınamayacaktır. Bu satır ile, asenkron metodun işleyişinin tamamlanması garanti altına alınmış olur. Uygulamanın bu kısmını aşağıdaki gibi daha kısa bir şekilde yazabiliriz.

```
double sonuc=t.EndInvoke(res);
Console.WriteLine(sonuc);
Console.ReadLine();
```

Burada direkt olarak EndInvoke metodu çağırılmış ve asenkron metodun dönüş değeri alınmıştır. Eğer bu noktada, asenkron metod halen daha tamamlanmamış ise, tamamlanıncaya kadar beklenir ve uygulama bu cevap gelinceye kadar duraksar.

Burada kullandığımız basit örnekteki teknik, uzak nesnelerin kullanıldığı Remoting uygulamaları içinde geçerlidir. Yine temsilciler, IAsyncResult arayüzü, BeginInvoke ve EndInvoke metodları bizim anahtar üyelerimiz olacaklardır. Remoting uygulamalarında, uzak nesneye ait metodların asenkron olarak nasıl çağırıldığını incelemeden önce, geçtiğimiz Remoting Makalesindeki UzakNesne.cs sınıfımıza aşağıdaki gibi yeni bir metod ekleyelim.

```
public double Alan(double yarıcap)
{
    Thread.Sleep(1500);
    return (yarıcap*3.14)/2;
}
```

Şimdi UzakNesne.cs dosyamızı yine aşağıdaki komut satırı ile derleyerek dll dosyamızı oluşturalım.

```
C:\ Visual Studio .NET 2003 Command Prompt

D:\Sunucu>csc /target:library uzaknesne.cs
Microsoft (R) Visual C# .NET Compiler version 7.10.3052.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

D:\Sunucu>dir
Volume in drive D is X-FILES
Volume Serial Number is E421-BC5C

Directory of D:\Sunucu

22.07.2004  01:27    <DIR>          .
22.07.2004  01:27    <DIR>          ..
30.06.2004  15:05           323 Sunucu.config
30.06.2004  15:08           314 Sunucu.cs
01.07.2004  00:22          3.072 Sunucu.exe
21.07.2004  23:55           736 UzakNesne.cs
22.07.2004  01:27          4.096 UzakNesne.dll
                5 File(s)              8.541 bytes
                2 Dir(s)        5.062.443.008 bytes free

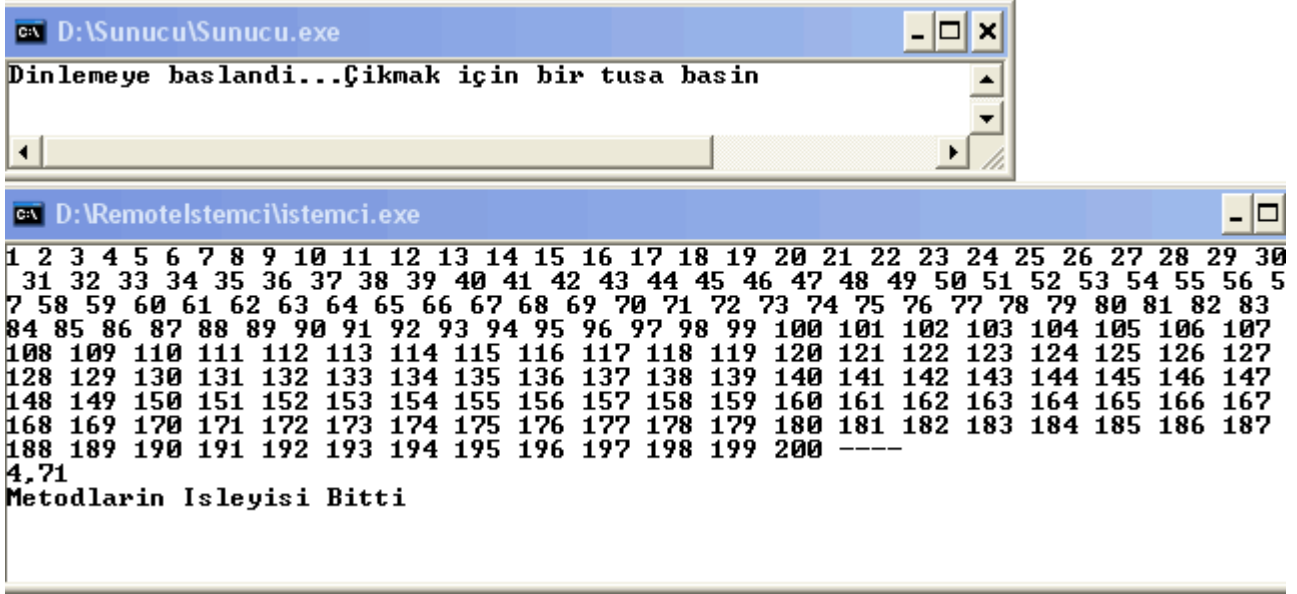
D:\Sunucu>
```

Geliştirdiğimiz istemci uygulamanın bulunduğu klasöre UzakNesne.dll dosyamızı kopyalayalım ve istemci sınıfımızın kodlarında aşağıdaki gibi yazalım. Bu örneğimizde, metodumuza senkron tekniği ile erişmekteyiz. Amacımız uzak nesne metoduna senkron olarak yani varsayılan yapısı ile eriştiğimizde, uygulamanın çalışma şeklini inceleyebilmek.

```
using System;
using System.Runtime.Remoting;

namespace istemciUygulama
{
    public class istemci
    {
        public static void Main(string[] args)
        {
            RemotingConfiguration.Configure("istemci.config");
            UzakNesne.Musteriler m=new UzakNesne.Musteriler();
            double alani=m.Alan(3);
            for(int i=1;i<=200;i++)
            {
                Console.Write(i.ToString()+" ");
            }
            Console.WriteLine("----");
            Console.WriteLine(alani);
            Console.WriteLine("Metodların İşleyisi Bitti");
            Console.ReadLine();
        }
    }
}
```

Remoting uygulamamızı bu haliyle denemek için önce sunucu uygulamamızı çalıştıralım ve sunucuya gelecek olan istemci taleplerini dinlemeye başlayalım. Daha sonra ise, istemci uygulamamızı çalıştıralım. Bu adımlardan sonra aşağıdakine benzer ekran görüntüsü ile karşılaşırız.



Bu ekran görüntüsünü açıklamadan anlamaya çalışmak bizim için yanıltıcı olacaktır. İstemci uygulama çalıştıktan yaklaşık 2 saniye kadar sonra bu ekran görüntüsü komple oluşacaktır. Yani for döngüsündeki kod satırları, uzak nesne üzerindeki metodumuzun işleyişi bitmeden ekrana yazılmayacaktır. Doğal olarak, uzak nesnedeki Alan metodunun istemci uygulamaya değer döndürmesiyle birlikte, for döngüsündeki işleyiş de ekrana yansıyacak ve tüm bu işlemler konsol penceresinde aynı zamanda gerçekleşecektir.

Oysaki uzak nesnemiz üzerindeki Alan metodunu asenkron olarak çağırırsaydık, öncelikle döngü içindeki kod satırları çalışarak ekrana 1 den 500' e kadar olan sayılar yazılacak ve bu işlemlerin ardından uzak nesnedeki metodun sonucu ekrana yazılacaktı. Dolayısıyla uzak nesnedeki Alan metodu çalıştırdıktan sonra, istemcideki izleyen kod satırları eş zamanlı olarak yürütülebilecekti. İşte bunu gerçekleştirebilmek için, istemci uygulamamızı aşağıdaki haliyle düzenlememiz ve uzak nesne metoduna asenkron olarak erişmemiz gerekmektedir.

```
using System;
```

```
using System.Runtime.Remoting;
```

```
namespace istemciUygulama
```

```
{
```

```
    public class istemci
```

```
    {
```

```
        private delegate double Temsilci(double d);
```

```
        public static void Main(string[] args)
```

```
        {
```

```
            RemotingConfiguration.Configure("istemci.config");
```

```
            UzakNesne.Musteriler m=new UzakNesne.Musteriler();
```

```
            Temsilci t=new Temsilci(m.Alan);
```

```
            IAsyncResult res=t.BeginInvoke(3,null,null);
```

```

for(int i=1;i<=500;i++)
{
    Console.Write(i.ToString()+" ");
}
Console.WriteLine("----");
double alani=t.EndInvoke(res);
Console.WriteLine(alani);
Console.WriteLine("Metodların İşleyisi Bitti");
Console.ReadLine();
}
}
}

```

Şimdi sunucuyu çalıştırıp, sonrada istemciyi çalıştırdığımızda sonuç olarak yine aynı ekran görüntüsünü elde ettiğimizi görürüz. Ancak bu kez döngü, asenkron metodun işleyişi ile birlikte çalışmış ve 1' den 500' e kadar olan sayıların konsol penceresine yazılmasının ardından kısa bir süre sonrada asenkron metodun sonucu ekrana yazılmıştır. İşleyiş tekniği ilk başta incelediğimiz örnekteki ile aynıdır. Bu kez sadece, temsilcimizi oluştururken, işaret edeceği metod olarak, istemci uygulamanın çalıştığı klasördeki dll içinde bulunan uzak nesne sınıf örneğine ait metod belirtilmiştir. Tabiki bu metodun uzak nesnedeki örneğinin çalıştırılması, konfigürasyon dosyasındaki kanal ayarlamaları sonucu ilgili kanal nesneleri üzerinden gerçekleşmektedir. Böylece, bir Remoting uygulamasında, uzak nesneler üzerindeki metodların asenkron olarak nasıl çalıştırılabileceğini en temel hatları ile ve yüzeysel olarak incelemiş olduk. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

CurrencyManager ile Navigasyon ve Temel Satır İşlemleri

Değerli Okurlarım, Merhabalar.

Bu makalemizde, CurrencyManager sınıfı yardımıyla, bağlantısız katman nesnelerinin işaret ettiği bellek bölgelerindeki veri satırları arasında navigasyon, satır ekleme, satır silme ve satır güncelleme işlemlerinin nasıl gerçekleştirildiğini incelemeye çalışacağız.

Genellikle bir veritabanı uygulamasını göz önüne aldığımızda, var olan satırlar arasında ileri ve geri yönlü hareketler sıklıkla kullanılan işlemler arasında yer almaktadır. Bir windows uygulamasında, bağlantısız katman nesnelerinin sahip olduğu veri kümelerini, form üzerinde yer alan çeşitli kontroller yardımıyla kullanıcıya sunarız. Ancak zaman zaman, bu veri kümesi içinde hareket ederken, veriye bağlı kontrollerinde güncel satıra ait alan bilgilerini göstermesini isteriz. Form üzerine yerleştirilen kontrollerin her biri, bu çeşit navigasyon işlemlerine izin veren CurrencyManager isimli nesne örneklerine sahiptir. Dolayısıyla navigasyon işlemlerinin bağlı kontrollere yansması için, bağlı kontrollere ait CurrencyManager nesnelerinin ilgili özelliklerinin kullanılması yeterlidir.

Bir windows uygulamasını düşündüğümüzde, Form üzerinde yer alan bağlı kontrollerinin tümünün aynı navigasyon hareketlerine izin vermesini istememiz son derece doğaldır. Bu zaten arzu edilen durumdur. Windows Form' larının BindingContext özelliği sayesinde, bağlı kontrollerin sahip olduğu CurrencyManager nesne örneklerini yönetebiliriz. Bu yönetim imkanı, satırlar arasında navigasyon, yeni kayıt ekleme, silme veya güncelleştirme gibi işlemleri yapabilme imkanına sahip olmamızı sağlamaktadır. Konuyu çok yüzeysel olarak anlattığımız bu kısa açıklamalardan sonra, olayı daha iyi ve net bir şekilde simüle edebilmek amacıyla aşağıdaki Form görüntüsüne sahip Windows Uygulamasını tasarlayarak işlemlerimize başlayalım.

Şekil 1. Form tasarımı.

Bu windows uygulamasında, Sql sunucusunda yer alan Northwind veritabanında yer alan Personel tablosuna ait veriler gösterilecektir. Başlangıç olarak, uygulama kodlarımız aşağıdaki gibidir.

```
SqlConnection con;
SqlDataAdapter da;
DataTable dt;
```

```
private void Bagla()
{
    lblPersonelID.DataBindings.Add("Text",dt,"PersonelID");
    txtPersonelAd.DataBindings.Add("Text",dt,"PersonelAd");
    txtPersonelSoyad.DataBindings.Add("Text",dt,"PersonelSoyad");
    txtSaatUcreti.DataBindings.Add("Text",dt,"SaatUcreti");
    txtCalismaSuresi.DataBindings.Add("Text",dt,"CalismaSuresi");
}
```

```
private void Form1_Load(object sender, System.EventArgs e)
{
    con=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=SSPI");
    da=new SqlDataAdapter("SELECT * FROM PERSONEL",con);
    dt=new DataTable("Personel");
    da.Fill(dt);
    Bagla();
}
```

Uygulama kodlarımıza kısaca baktığımızda, Sql sunucusuna bağlanarak Northwind veritabanındaki Personel tablosuna ait tüm satırları, bir DataTable' a aktardığımızı ve Form üzerindeki kontrolleride bu veri kaynağına bağladığımızı görürüz. Kontrollerimizi, tablonun ilgili alanlarına bağlarken DataBindings özelliğinin Add metodundan yararlanmaktayız.

Add metodu ilk parametre olarak, veri kaynağındaki ilgili verinin, kontrolün hangi özelliğine bağlanacağını belirtir. İkinci parametrede bu ilk parametredeki özellik için hangi veri kaynağının kullanılacağı belirtilir. Son olarak üçüncü parametrede ise, ilk parametrede belirtilen kontrol özelliğine, ikinci parametredeki veri kaynağının hangi alanının bağlanacağı belirlenir. Böylece, DataTable nesneminin bellekte işaret ettiği bölgedeki satırlara ait alanlar, Form üzerindeki kontrollere bağlanmış olacaktır. Uygulamamızı bu haliyle derleyip çalıştırdığımızda, veri kümesindeki ilk satıra ait bilgilerin Form üzerindeki kontrollere eklendiğini görürüz.

Şekil 2. Uygulamanın Çalışması.

Şimdi ilk yapmak istediğimiz, veri kümesindeki satırlar arasında hareket edebilme imkanına sahip olmaktır. Şu aşamda 4 temel hareketimiz olabilir. Sonraki satıra geçiş, önceki satıra geçiş, son satıra geçiş ve ilk satıra geçiş. Bu işlemleri gerçekleştirebilmek için, Form' umuza ait BindingContext özelliğini kullanacağız. Şimdi formumuza 4 adet button kontrolü ekleyelim ve uygulama kodlarımızı aşağıdaki şekilde geliştirelim.

```
SqlConnection con;
SqlDataAdapter da;
DataTable dt;
CurrencyManager cm;
```

```
private void Bagla()
{
    lblPersonelID.DataBindings.Add("Text",dt,"PersonelID");
    txtPersonelAd.DataBindings.Add("Text",dt,"PersonelAd");
    txtPersonelSoyad.DataBindings.Add("Text",dt,"PersonelSoyad");
    txtSaatUcreti.DataBindings.Add("Text",dt,"SaatUcreti");
```

```

txtCalismaSuresi.DataBindings.Add("Text",dt,"CalismaSuresi");
}

private void Form1_Load(object sender, System.EventArgs e)
{
    con=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=SSPI");
    da=new SqlDataAdapter("SELECT * FROM PERSONEL",con);
    dt=new DataTable("Personel");
    da.Fill(dt);
    Bagla();
    cm=(CurrencyManager)this.BindingContext[dt];
    lblPozisyon.Text="Guncel Pozisyon -----> "+Convert.ToString(cm.Position+1);
}

private void btnIlkSatiraGotur_Click(object sender, System.EventArgs e)
{
    cm.Position=0;
    lblPozisyon.Text="Guncel Pozisyon -----> "+Convert.ToString(cm.Position+1);
}

private void btnOncekiSatiraGotur_Click(object sender, System.EventArgs e)
{
    cm.Position--;
    lblPozisyon.Text="Guncel Pozisyon -----> "+Convert.ToString(cm.Position+1);
}

private void btnSonrakiSatiraGotur_Click(object sender, System.EventArgs e)
{
    cm.Position++;
    lblPozisyon.Text="Guncel Pozisyon -----> "+Convert.ToString(cm.Position+1);
}

private void btnSonSatiraGotur_Click(object sender, System.EventArgs e)
{
    cm.Position=dt.Rows.Count-1;
    lblPozisyon.Text="Guncel Pozisyon -----> "+Convert.ToString(cm.Position+1);
}

```

Uygulama kodlarımızda en önemli nokta, Form üzerindeki kontrollerin sahip oldukları CurrencyManager nesnelerini elde edebilmek için, Form' a ait BindingContext özelliğinden yararlanmamızdır.

```
cm=(CurrencyManager)this.BindingContext[dt];
```

Bu satır ile, DataTable veri kaynağına için bir CurrencyManager nesnesi oluşturulur. Artık navigasyon işlemi için tek yapmamız gereken, CurrencyManager sınıfına ait Position özelliğinin kullanılmasıdır. Bu özelliğin değişmesi durumunda, Form üzerinde yer alan ve DataTable' a ait veri kaynağına bağlı olan tüm kontrollerin içeriği güncel satıra ait verileri gösterecek şekilde değişecektir. Örneğin, bir satır ileri gitmek için Position özelliğinin değerini 1 arttırmamız yeterli olurken, son satıra gitmek için, veri kaynağının sahip olduğu toplam satır sayısını 1 eksiltiriz.

```
cm.Position++;
```

```
.  
.   
.
```

```
cm.Position=dt.Rows.Count-1;
```

Uygulamamızı çalıştırdığımızda, satırlar arasında istediğimiz şekilde gezebildiğimizi görürüz.



Şekil 3. Navigasyon.

CurrencyManager sınıfı yardımıyla yapabileceklerimiz sadece Navigasyon işlemleri ile sınırlı değildir. Ayrıca, satır ekleme, satır silme veya satır güncelleştirme gibi temel tablo işlemlerinde gerçekleştirebiliriz. Yeni bir satır eklemek için CurrencyManager sınıfına ait, aşağıda prototipi verilen AddNew metodunu kullanırız.

```
public override void AddNew();
```

Uygulamamızda yeni bir satır eklemek için öncelikle aşağıdaki kodları yazalım.

```
private void Temizle()  
{  
    lblPersonelID.Text="";
```



```

for(int i=0;i<this.Controls.Count;i++)
{
    if(this.Controls[i] is TextBox)
    {
        this.Controls[i].Text="";
    }
}
}

```

```

private void btnYeniSatirEkle_Click(object sender, System.EventArgs e)
{
    cm.AddNew();
    Temizle();
}

```

```

private void btnVeritabaninaYaz_Click(object sender, System.EventArgs e)
{
    SqlCommandBuilder cmb=new SqlCommandBuilder(da);
    da.Update(dt);
}

```

Burada, önemli olan AddNew metodunun kullanılışından sonra, açılan yeni satırın o anki veri kümesinde yerini alması için ileri veya geri yönlü bir hareket yapılmasının yeterli olduğuudur. Elbette, CurrencyManager yardımıyla, güncel veri kümesine eklenen satırların veritabanınada yansıtılabilmesi için, SqlDataAdapter nesnemize ait Update metodunun çalıştırılması gerekmektedir. Güncel satırlar üzerinde değişiklik yapıldığında bu değişikliklerin onaylanabilmesi ve veri kümesine yansiyabilmesi için ya satırlar arasında hareket edilmesi yada aşağıda prototipi verilen EndCurrentEdit metodunun uygulanması gerekir.

public override void EndCurrentEdit();

Veri kümesinden herhangi bir satırı silmek istediğimizde ise, aşağıdaki prototipe sahip olan RemoveAt metodunu kullanabiliriz.

public override void RemoveAt(int index);

Bu metod parametre olarak, silinecek satırın indeksini almalıdır. Satır güncelleme ve silme işlemleri için ilgili kodlarımız ise aşağıdaki gibidir.

```

private void btnDegisiklikleriKaydet_Click(object sender, System.EventArgs e)
{
    if(MessageBox.Show("Değişiklikler kaydedilsin mi?", "Değişiklik" , MessageBoxButtons.OKCancel, MessageBoxIcon.Question)
    == DialogResult.OK)
    {

```

```

cm.EndCurrentEdit();
}
else
{
    cm.CancelCurrentEdit();
}
}

```

```

private void btnGuncelSatiriSil_Click(object sender, System.EventArgs e)
{
    cm.RemoveAt(cm.Position);
}

```

Elbette yaptığımız satır ekleme, güncelleme işlemlerini iptal etmek istersek, prototipi aşağıdaki gibi olan CancelCurrentEdit metodunu kullanabiliriz. Bu metod ile o anda yapılan yeni satır ekleme veya güncelleme işleminin oluşturduğu değişikliği geri alabiliriz. Tabiki satırdaki değişiklik EndCurrentEdit metodu ile yada navigasyon hareketi ile onaylanmadıysa. Bu durum sadece satır silme işlemi için geçerli değildir. Dolayısıyla satır silindiğinde, CancelCurrentEdit bu işlemi geri almaz.

public override void CancelCurrentEdit();

CurrencyManager sınıfının işimize yarayacak bir kaç olayı da vardır. Bunlardan birisi PositionChanged olayıdır. Bu olay güncel satır pozisyonu değiştiğinde çalışmaktadır. Uygulamamızdaki pozisyon bilgisini lblPozisyon kontrolünde göstermek için kullandığımız kod satırını bu olay içine yerleştirebiliriz. Elbette öncelikle, CurrencyManager nesnemiz için aşağıdaki kod satırı ile, PositionChanged olayını eklememiz gerekmektedir.

```

cm.PositionChanged+=new EventHandler(cm_PositionChanged);
.
.
.
private void cm_PositionChanged(object sender, EventArgs e)
{
    lblPozisyon.Text="Guncel Pozisyon -----> "+Convert.ToString(cm.Position+1);
}

```

Bu makalemiz ile, CurrencyManager sınıfının temel metodlarını ve özelliklerini inceleyerek, veri kümeleri üzerinde navigasyon, ekleme, güncelleme ve silme işlemlerinin nasıl yapıldığını öğrenmeye çalıştık. Umuyorum ki faydalı bilgiler verebilmişimdir. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Identity Değerlerinin Çalışma Zamanında Elde Edilmesi

Değerli Okurlarım, Merhabalar.

Bu makalemizde, veritabanlarında otomatik olarak artan identity alanlarının değerlerinin, çalışma zamanında uygulama ortamlarına nasıl yansıtılabileceğini incelemeye çalışacağız. Çoğunlukla, tablolarımızda yer alan satırların birbirlerinden kolayca ayırt edilebilmelerini sağlamak için, primary key alanlarını kullanırız. Genellikle bu alanları otomatik olarak artan sayısal değerler üzerinde yapılandırırız. Örnek olarak aşağıdaki tabloyu göz önüne aldığımızda, PersonelID alanının 1 sayısal değerinden başlayarak 1'er artan ve primary key özelliğine sahip olduğunu görürüz.

Column Name	Data Type	Length
PersonelID	int	4
PersonelAd	nvarchar	50
PersonelSoyad	nvarchar	50
SaatUcreti	decimal	9
CalismaSuresi	decimal	9

Description	
Default Value	
Precision	10
Scale	0
Identity	Yes
Identity Seed	1
Identity Increment	1
Is RowGuid	No
Formula	
Collation	

Şekil 1. PersonelID alanının özellikleri

Bu alanın değeri, tablonun bulunduğu sql sunucusu tarafından otomatik olarak arttırılmaktadır. Buraya kadar her şey zaten bildiğimiz olaylardır. Sorun, bir uygulamada bu tabloya yeni bir alan eklendiğinde, o an veritabanı tarafından otomatik olarak arttırılan alanın değerinin ortama yansıtılmasında ortaya çıkmaktadır. Nitekim bu alanın yeni oluşan değerini elde etmek için, ilgili satırın uygulama ortamına yeniden çekilmesi gerekmektedir. Bu ise özellikle tabloya ait tüm satırların çekildiği durumlarda gereksiz zaman kaybının yaşanmasına neden olmaktadır. Oysaki uygulayacağımız basit kodlamalar ile, yeni eklenen satırlara ait otomatik artan değerleri uygulama ortamına, minimum eforu sarfederek kolaylıkla alabiliriz. Sorunu daha iyi anlayabilmek amacıyla basit bir windows uygulaması geliştirelim. Form görüntümüz aşağıdakine benzer olmalıdır.

Şekil 2. Uygulama Formumuz.

Uygulamamız basit olarak, Sql sunucusunda yer alan Northwind veritabanındaki Personel tablosuna ait verileri çekmekte ve yeni satırlar eklenmesine izin vermektedir. Uygulamamıza ait ilk kodlar ise aşağıdaki gibidir.

```
SqlConnection con;
SqlDataAdapter da;
DataTable dt;
```

```
private void btnVeriCek_Click(object sender, System.EventArgs e)
{
    con=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=SSPI");
    da=new SqlDataAdapter("SELECT * FROM PERSONEL",con);
    dt=new DataTable("Personel");
    da.Fill(dt);
    dataGrid1.DataSource=dt;
}
```

```
private void btnEkle_Click(object sender, System.EventArgs e)
{
    DataRow dr;
    dr=dt.NewRow();
    dr[1]=txtPersonelAd.Text;
    dr[2]=txtPersonelSoyad.Text;
```

```

dr[3]=txtSaatUcreti.Text;
dr[4]=txtCalismaSuresi.Text;
dt.Rows.Add(dr);

SqlCommandBuilder cmb=new SqlCommandBuilder(da);
da.Update(dt);
}

```

Şimdi uygulamamızı çalıştıralım, Veri Çek başlıklı butona basalım ve dataGrid kontrolümüzün Personel verileri ile dolduğunu görelim. Yeni bir personel kaydı girdiğimizde ve bu bilgileri Ekle butonuna basarak veritabanına gönderdiğimizde ekran görüntümüzün aşağıdaki gibi oluştuğunu farkedebiliriz.

The screenshot shows a Windows application titled "Identity". It has two main panels. The left panel, titled "Veri Çek", contains a data grid with the following data:

	PersonelID	PersonelAd	PersonelSo	SaatUc
	58	Ahmet Faruk	Nacaroglu	1
	59	Sefer	Algan	1
▶	78	Burak Selim	Senyurt	1
	79	Okan	Barlas	1
	80	Feryat	Olca	1
	81	Resul	Çavusoglu	1
	(null)	Bil	Geyts	25
*				

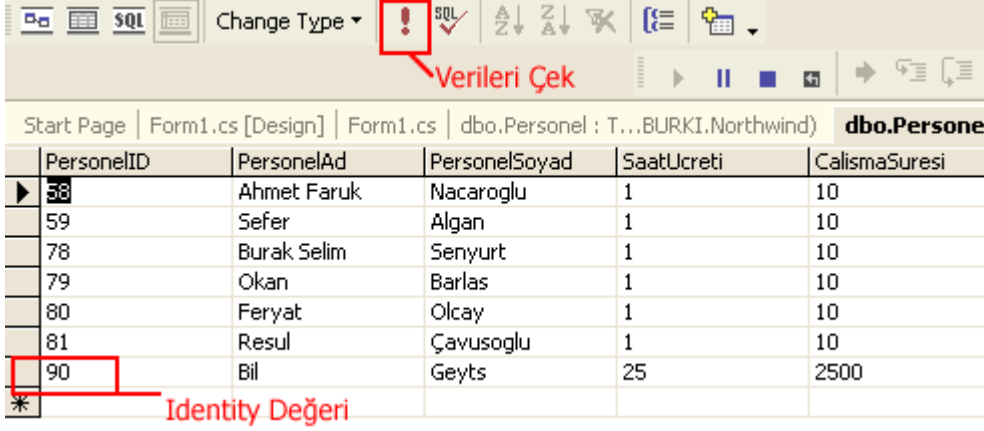
The right panel, titled "Yeni Personel Girişi", contains a form with the following fields:

- Ad: Bil
- Soyad: Geyts
- Saat Ücreti: 25
- Çalışma Süresi: 2500

At the bottom of the right panel is a button labeled "Ekle".

Şekil 3. Yeni satır eklenişi ve Identity alanının değeri.

Dikkat edilecek olursa, yeni satırımız bağlantısız katman nesnemiz olan DataTable'ın Rows koleksiyonuna ve ayrıca sql sunucusu üzerindeki Personel tablosuna başarılı bir şekilde eklenmiştir. Sorun şudurki, Veri Çek butonuna basmadığımız ve Personel tablosuna ait verileri DataTable nesnesine tekrardan doldurmamız süreci, PersonelID alanının veritabanı tarafından otomatik olarak atanan değerini göremeyiz. Oysaki böyle bir değer, gerçek bazlı bir projede başka bir işlem için veri olarak kullanılmak istenebilir. Örneğin, bir call center'daki operatör, müşterisi için yeni açtığı bir dosyaya ait Identity değerini, dosya referans numarası olarak vermek durumunda olabilir. Kaldık, eklediğimiz satır için geçerli olan Identity değeri otomatik olarak oluşturulmuştur. Ancak bağlantısız katman nesnesinin henüz bundan haberi yoktur. Bu noktada programı sonlandırmadan Vs.Net ortamında, Personel tablosu bilgilerine bakıldığında, sql sorgusunu yeniden execute edersek Identity değerinin oluşturulmuş olduğunu gözlemleyebiliriz.



PersonelID	PersonelAd	PersonelSoyad	SaatUcreti	CalismaSuresi
58	Ahmet Faruk	Nacaroglu	1	10
59	Sefer	Algan	1	10
78	Burak Selim	Senyurt	1	10
79	Okan	Barlas	1	10
80	Feryat	Olçay	1	10
81	Resul	Çavusoglu	1	10
90	Bil	Geyts	25	2500

Şekil 4. Identity değeri.

İşte makalemize konu olan sorun bu alanın değerinin, satır veritabanına eklendiği zaman nasıl uygulama ortamına alınacağıdır. İki alternatif çözüm yolumuz vardır. Bunlardan birincisi, SqlDataAdapter sınıfının RowUpdated olayının kullanılmasıdır. Diğer alternatif yol ise, yeni identity değerini uygulama ortamına çekebileceğimiz bir Stored Procedure yardımıyla, satır ekleme işleminin yapılmasıdır. Hangi yolu seçersek seçelim ikisinde ortak bir anahtar sözcüğü kullanmaktadır. Bu anahtar sözcük sql' e ait olan @@IDENTITY 'dir. Bu anahtar sözcük, eklenen satıra ait, veritabanı tarafından otomatik olarak üretilen identity değerini temsil etmektedir. İlk olarak RowUpdated olayı ile bu işin nasıl sağlanacağına bakalım. Tek yapmamız gereken SqlDataAdapter nesnemize, RowUpdated olayını eklemek ve bu olay metodunu kodlamaktır.

```
private void btnVeriCek_Click(object sender, System.EventArgs e)
{
    .
    .
    .
    da.RowUpdated+=new SqlRowUpdatedEventHandler(da_RowUpdated);
}
```

```
private void btnEkle_Click(object sender, System.EventArgs e)
{
    .
    .
    .
}
```

```
private void da_RowUpdated(object sender, SqlRowUpdatedEventArgs e)
{
    SqlCommand cmd=new SqlCommand("SELECT @@IDENTITY",con);
    if((e.Status==UpdateStatus.Continue) && (e.StatementType==StatementType.Insert))
```

```

{
    e.Row["PersonelID"]=cmd.ExecuteScalar();
    e.Row.AcceptChanges();
}
}

```

Burada RowUpdated metodunda, veritabanına yeni bir satır eklendiğinde, @@IDENTITY değerini alacak sorguyu çalıştıran bir SqlCommand nesnesi kullanılmaktadır. Böylece, bağlantısız katmana ait yeni eklenen satırın ilgili alanına (yani PersonelID alanına), sorgunun sonucunu alabiliriz. Uygulamamızı bu haliyle çalıştırdığımızda ve yeni bir satır eklediğimizde, PersonelID alanı için veritabanı tarafından üretilen otomatik değerinde, uygulama ortamına alındığını gözlemleriz.

The screenshot shows a Windows application titled "Identity". It has two main panes. The left pane, titled "Veri Çek", contains a table with the following data:

	PersonelID	PersonelAd	PersonelSo	SaatUcre
▶	58	Ahmet Faru	Nacaroglu	1
	59	Sefer	Algan	1
	78	Burak Seli	Senyurt	1
	79	Okan	Barlas	1
	80	Feryat	Olçay	1
	81	Resul	Çavusoglu	1
	90	Bil	Geyts	25
	91	Jon	Finli	35
*				

The row with ID 91 is highlighted with a red box. The right pane, titled "Yeni Personel Girişi", contains a form with the following fields:

- Ad: Jon
- Soyad: Finli
- Saat Ücreti: 35
- Çalışma Süresi: 4500

At the bottom of the right pane is a button labeled "Ekle".

Şekil 5. Identity değerinin elde edilmesi.

Gelelim ikinci yola. İkinci yolumuz ise, ekleme işleminin bir Stored Procedure yardımıyla yapılmasıdır. Bu Stored Procedure, ekleme işlemini yaparken, @@IDENTITY değerini de ortama bir Output parametresi vasıtasıyla gönderecektir. Bu amaçla Personel tablomuz için aşağıdaki Stored Procedure' ü oluşturalım.

```

ALTER PROCEDURE dbo.PersonelEkle
(
    @PersonelID int OUTPUT,
    @PersonelAd varchar(50),
    @PersonelSoyad varchar(50),
    @SaatUcreti decimal,
    @CalismaSuresi decimal
)

```

AS

```
INSERT INTO Personel (PersonelAd,PersonelSoyad,SaatUcreti,CalismaSuresi) VALUES
(@PersonelAd,@PersonelSoyad,@SaatUcreti,@CalismaSuresi)
```

```
SELECT @PersonelID=@@IDENTITY
```

RETURN

Bu Stored Procedure' de en önemli nokta Select sorgusunda @@IDENTITY değerinin, bir Output parametresine aktarılmış olmasıdır. Bu sayede, SP' yi çalıştırdığımızda, veri tabanı tarafından oluşturulan otomatik değeri, uygulama ortamımızda elde edebiliriz. Tek yapmamız gereken aşağıdaki kodları yazmaktır.

```
private void btnSPileEkle_Click(object sender, System.EventArgs e)
{
    SqlCommand cmd=new SqlCommand("dbo.PersonelEkle",con);
    cmd.CommandType=CommandType.StoredProcedure;

    cmd.Parameters.Add("@PersonelID",SqlDbType.Int);
    cmd.Parameters.Add("@PersonelAd",SqlDbType.VarChar,50);
    cmd.Parameters.Add("@PersonelSoyad",SqlDbType.VarChar,50);
    cmd.Parameters.Add("@SaatUcreti",SqlDbType.Decimal);
    cmd.Parameters.Add("@CalismaSuresi",SqlDbType.Decimal);

    cmd.Parameters["@PersonelID"].Direction=ParameterDirection.Output;

    cmd.Parameters["@PersonelAd"].Value=txtPersonelAd.Text.ToString();
    cmd.Parameters["@PersonelSoyad"].Value=txtPersonelSoyad.Text.ToString();
    cmd.Parameters["@SaatUcreti"].Value=Convert.ToDecimal(txtSaatUcreti.Text);
    cmd.Parameters["@CalismaSuresi"].Value=Convert.ToDecimal(txtCalismaSuresi.Text);

    con.Open();
    cmd.ExecuteNonQuery();

    /* Yeni satırı DataTable' ımızda ekliyoruz.*/
    DataRow dr;
    dr=dt.NewRow();
    dr[0]=cmd.Parameters["@PersonelID"].Value; /* Veritabanında henüz oluşturulan otomatik değeri alıp, satırın PersonelID alanına aktarıyoruz.*/
    dr[1]=txtPersonelAd.Text;
    dr[2]=txtPersonelSoyad.Text;
```



```

dr[3]=txtSaatUcreti.Text;
dr[4]=txtCalismaSuresi.Text;
dt.Rows.Add(dr);
dt.AcceptChanges();
}

```

Uygulamamızı bu haliyle çalıştırdığımızda yeni eklenen satır için, veritabanı tarafından otomatik olarak arttırılan identity değerinin kolayca elde edilebildiğini görürüz.

The screenshot shows a Windows application titled 'Identity'. It has two main panes. The left pane, labeled 'Veri Çek', contains a table with the following data:

	PersonelID	PersonelAd	PersonelSo	SaatUcre
▶	58	Ahmet Faru	Nacaroglu	1
	59	Sefer	Algan	1
	78	Burak Seli	Senyurt	1
	79	Okan	Barlas	1
	80	Feryat	Olçay	1
	81	Resul	Çavusoglu	1
	90	Bil	Geyts	25
	97	Maykıl	Del	25
*				

The right pane, labeled 'Yeni Personel Girisi', contains a form with the following fields and values:

- Ad: Maykıl
- Soyad: Del
- Saat Ücreti: 25
- Çalışma Süresi: 120

At the bottom of the right pane are two buttons: 'Ekle' and 'SP İle Ekle'.

Şekil 6. SP ile Identity değerinin elde edilmesi.

Gelelim bu iki seçenekten hangisinin tercih edileceğine. Microsoft' un bu konuda yaptığı testlere göre, Stored Procedure yardımıyla Identity değerlerinin elde edilmesi, RowUpdated olayının kullanıldığı tekniğe nazaran daha performanslı ve verimli. Dolayısıyla SP kullanımını tercih etmek daha doğru bir seçenek gibi gözükmemektedir. Bir başka konu ise, Access tipi tablolar için aynı senaryonun nasıl işleyeceğidir. Access tipi tablolarda, Output parametresi desteklenmediğinden, SP kullanımı ile identity değerinin elde edilmesi gerçekleşmeyecektir. Bu nedenle tek seçenek, RowUpdated olayında @@IDENTITY değerinin elde edilmesidir. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinizde mutlu günler dilerim.

Localization (Yerelleştirme) – 1

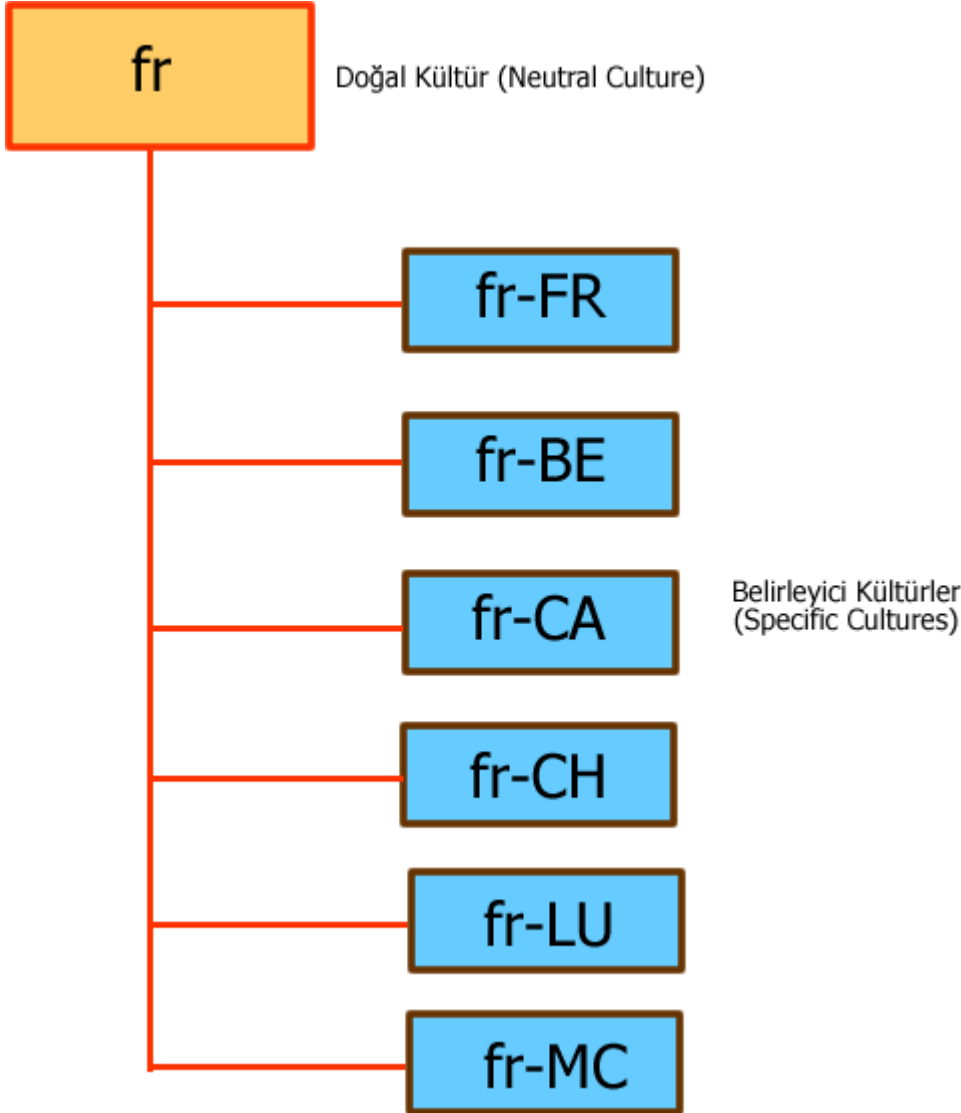
Değerli Okurlarım, Merhabalar.

Dünya çapında yada başka bir deyişle global çapta uygulamalar geliştirilirken karşılaşılabilecek zorluklardan birisi, uygulamanın farklı kültür ve dil seçeneklerine göre çalışabilecek şekilde tasarlanmasıdır. Eski programlama dilleri göz önüne alındığında, özellikle farklı dil desteği sağlayacak uygulamaların geliştirilmesi tam anlamıyla bir kabus olmuştur. Söz gelimi 2 dile destek verecek bir uygulama geliştirilmek istendiğinde, her iki dil içinde farklı uygulama kodları yazılması gerekirdi. Böyle bir durumda, uygulamanın piyasaya sürülmesinden sonra yapılacak güncelleme paketleri içinde aynı durum söz konusuydu. Dahası, 3ncü bir dilin desteğinin sağlanması için, aynı projenin bu kezde bu dil için geliştirilmesi gerekmektedir. Dil çeşitliliğinin yanı sıra, aynı dili konuşan fakat farklı takvimler, farklı parasal formatlar, farklı sayısal formatlar hatta farklı sıralamalar kullanan kültürler için içine sokulduğunda durum tam anlamıyla bir paradoks haline gelmektedir.

Peki Microsoft .Net ile bu soruna nasıl bir çözüm getirmektedir? İşte bu günkü makalemizin konusu budur. Microsoft .Net platformu içinde geliştirdiğimiz çözümlerin (solutions), çeşitli dillere veya kültürlere destek verebilmesini sağlamak amacıyla, öncekilerine nazaran çok daha mantıklı ve verimli bir teknik uygulanır. Bu teknikte, uygulama kodu bir kez yazılır. Bu uygulamanın dil desteği ve kültür desteği vermesi için ise, framework içinde yer alan System.Globalization isim alanındaki sınıflardan ve resource files dediğimiz kaynak dosyalarından yararlanılır. Bir başka deyişle, bir uygulamanın yerelleştirilmesi yani dil desteği ve kültürel desteğin sağlanması için, uygulama kodları ile destek birimleri birbirinden ayrı tutulmuştur. Böylece, bir uygulamayı n sayıda kültür için yerelleştirmek mümkün olmaktadır.

Bu teorik bilgiler ilk okunduğunda akla karmaşık gelebilir. Olayı daha iyi anlayabilmek için, yeryüzünde yer alan toplumların birbirlerinden farklılaşmasına neden olan kültürel özellikleri keşfetmek gerekmektedir. Dünyamız konuşulan ana dillere göre çeşitli bölgelere ayrılmıştır. Bu diller, bir toplumun doğal kültürünü (Neutral Culture) temsil eder. Yada başka bir deyişle doğal kültürler dillere göre kategorize edilir. Örneğin İngilizce, Almanca, Fransızca, İtalyanca dilleri doğal kültürleri (Neutral Cultures) yansıtmaktadır. Bununla birlikte, çeşitli dillere destek verecek çevirisel uygulamalarda, doğal kültürlerin bilinmesi yeterli iken, parasal işlemlerin, sayısal işlemlerin, tarihsel işlemlerin, sıralamaların vb... yer aldığı daha belirleyici yerelleştirme işlemlerinde, dile bağlı doğal kültür bilgileri yeterli değildir. Örneğin, İngilizce konuşan pek çok alt kültür vardır. İngiltere, Amerika, Kanada, Avusturalya vb... Bu ülkelerin her biri İngilizce konuşmakla birlikte, farklı kültür özelliklerine sahip olabilirler. Örneğin kullandıkları para birimi farklılığı, zip kodların farklılığı, sayısal ifadelerdeki nokta ve virgöl kullanımlarının farklılıkları vb... Dolayısıyla bir uygulamayı, belirli bir kültür için özelleştirecek isek, Specific Culture denilen alt kültür bilgilerinden yararlanmamız gerekmektedir.

Tabiki kültür bilgilerinin belirli bir standart çerçevesinde ifade edilmesi ve küresel çapta kabul görmesi oldukça önemlidir. Microsoft bunun için, IETF tarafından belirlenmiş RFC 1766 standardını kullanır. Bu standarda göre, her doğal kültür ve bu doğal kültüre bağlı alt kültürler ikiye karakterden oluşan kısaltma kodlar ile ifade edilirler. Tabiki bir doğal kültür birden fazla alt kültüre ve bu alt kültürlerde başka alt kültürlere sahip olabilirler. Konuyu aşağıdaki örnek şekil daha iyi ifade edecektir.



Şekil 1. Fransızca'nın alt kültürleri.

Örneğin Fransızca dili bir doğal kültür olarak ele alındığında, dünya üzerinde Fransızca'yı konuşan bölgelerde bu kültürün alt kültürlerini oluşturur. fr-FR Fransa'yı, fr-BE Belçikayı fr-LU Lüksemburg'u temsil eden belirleyici kültür kodlarıdır. Peki uygulamalarımızda bu kültür kodlarını nasıl kullanabiliriz? Bunun için System.Globalization isim alanındaki CultureInfo ve RegionInfo sınıflarını kullanabiliriz. CultureInfo sınıfı yardımıyla belirli bir kültüre ait bilgilere sahip oluruz. RegionInfo sınıfı ile de, belirli bir bölgeye ait bilgilere ulaşırız. Örneğin para birimi ve metrik sistemin kullanılıp kullanılmadığı gibi bilgiler RegionInfo sınıfını ilgilendirirken, takvim bilgisi, doğal ad gibi bilgilerde CultureInfo sınıfını ilgilendirir. İşin özel yanı, bir uygulamanın kullanıcı arayüzündeki yerel kriterleri değiştirmek istediğimizde, uygulamanın çalıştığı thread nesnesine ait metodları kullanamamızın yeterli olacaktır. Tabiki burada bahsi geçen proses, belirtilen bir kültür bilgisine yani CultureInfo nesnesine göre ayarlandığı takdirde, programın yerelleştirilmesi gerçekleştirilmiş olacaktır.

Şimdi bu bahsettiklerimizi daha iyi anlayabileceğimiz basit bir uygulama geliştirmeye çalışalım. Bu uygulamamızda, kabul gören standartlar içindeki tüm kültürlere ait kodları listeleyeceğimiz bir treeView kontrolü kullanacağız. Her hangibir kültür seçildiğinde bu kültüre ait bilgiler ekrandaki kontrollere dolacak. Ayrıca, tarih, saat ve parasal bilgi veren textBox' larımızda örnek değerler tutacak

ve bu bilgilerin seçilen kültürün belirttiği özelliklere göre gösterilmesi sağlanacak. Öncelikle Vs.Net içinde aşağıdaki forma görüntüsüne sahip olan bir windows uygulaması oluşturalım.

Şekil 2. Form tasarımıımız.

Gelelim uygulama kodlarımıza. Her şeyden önce System.Globalization isim alanını uygulamamıza using anahtar kelimesi ile eklemeliyiz. Nitekim kullancanağımız CultureInfo ve RegionInfo sınıfları bu isim alanı içinde yer almaktadır. Kulturler başlıklı butona bastığımızda, Microsoft' un kullandığı standart içindeki tüm kültür kodlarının listesini almak ve bunları treeView kontrolünde ağaç yapısı şeklinde hiyerarşik bir düzende göstermek istiyoruz. İşte Kulturler başlıklı butona ait kodlarımız.

```
private void btnKulturler_Click(object sender, System.EventArgs e)
{
    CultureInfo[] kulturDizi=CultureInfo.GetCultures(CultureTypes.AllCultures);

    TreeNode[] nodes=new TreeNode[kulturDizi.Length];

    int i=0;
    TreeNode parent=null;
```

```

foreach(CultureInfo kulturBilgi in kulturDizi)
{
    nodes[i]=new TreeNode();
    nodes[i].Text=kulturBilgi.DisplayName;
    nodes[i].Tag=kulturBilgi;

    if(kulturBilgi.IsNeutralCulture)
    {
        parent=nodes[i];
        treeView1.Nodes.Add(nodes[i]);
    }
    else if(kulturBilgi.ThreeLetterISOLanguageName==CultureInfo.InvariantCulture.ThreeLetterISOLanguageName)
    {
        treeView1.Nodes.Add(nodes[i]);
    }
    else
    {
        parent.Nodes.Add(nodes[i]);
    }
}
}

```

Öncelike, var olan kültürleri elde etmek için CultureInfo sınıfına ait GetCultures metodunu kullandık. Bu metoda parametre olarak, CultureTypes.AllCultures değerini verdik. Metodumuz bu parametre değeri dışında aşağıdaki değerleride alabilmektedir.

CultureTypes Numaralandırıcı Değeri	Açıklama
AllCultures	Tüm kültürleri listeler.
InstalledWin32Cultures	Windows sistemlerinde kullanılan kültürleri listeler.
NeutralCultures	Sadece dille ilişkilendirilmiş doğal kültürleri listeler.
SpecificCultures	Belirleyici kültürleri listeler.

Bundan sonraki kod satırlarında treeView kontrolümüzü doldurmak için gerekli işlemler yapılmıştır. Dikkat etmemiz gereken noktalar, foreach döngüsü içindeki if koşullarıdır. Buradaki ilk koşulda CultureInfo nesnesinin doğal bir kültür olup olmadığına bakılır. Diğer koşulda ise kültürün invariant culture olup olmadığı koşulu değerlendirilir. Invariant kültürler, gerçek kültürlerden bağımsız olan yapılardır. Şimdi uygulamamızın diğer kodlarını oluşturalım.

```

private void Temizle()
{
    for(int i=0;i<this.Controls.Count;i++)

```

```

{
    if(this.Controls[i] is TextBox)
    {
        this.Controls[i].Text="";
    }
}
}

private void treeView1_AfterSelect(object sender, System.Windows.Forms.TreeViewEventArgs e)
{
    Temizle();
    CultureInfo guncelKultur=(CultureInfo)treeView1.SelectedNode.Tag;
    txtName.Text=guncelKultur.Name;
    txtNativeName.Text=guncelKultur.NativeName;
    txtEnglishName.Text=guncelKultur.EnglishName;

    if(!guncelKultur.IsNeutralCulture)
    {
        RegionInfo bolgeBilgisi=new RegionInfo(guncelKultur.LCID);
        txtCurrency.Text=bolgeBilgisi.CurrencySymbol;
        txtRegion.Text=bolgeBilgisi.DisplayName;
        txtMetric.Text=bolgeBilgisi.IsMetric.ToString();

        Thread.CurrentThread.CurrentCulture=guncelKultur;
        double sayi=4587512.451;
        txtNumber.Text=sayi.ToString("N");

        txtDate.Text=DateTime.Today.ToString("D");

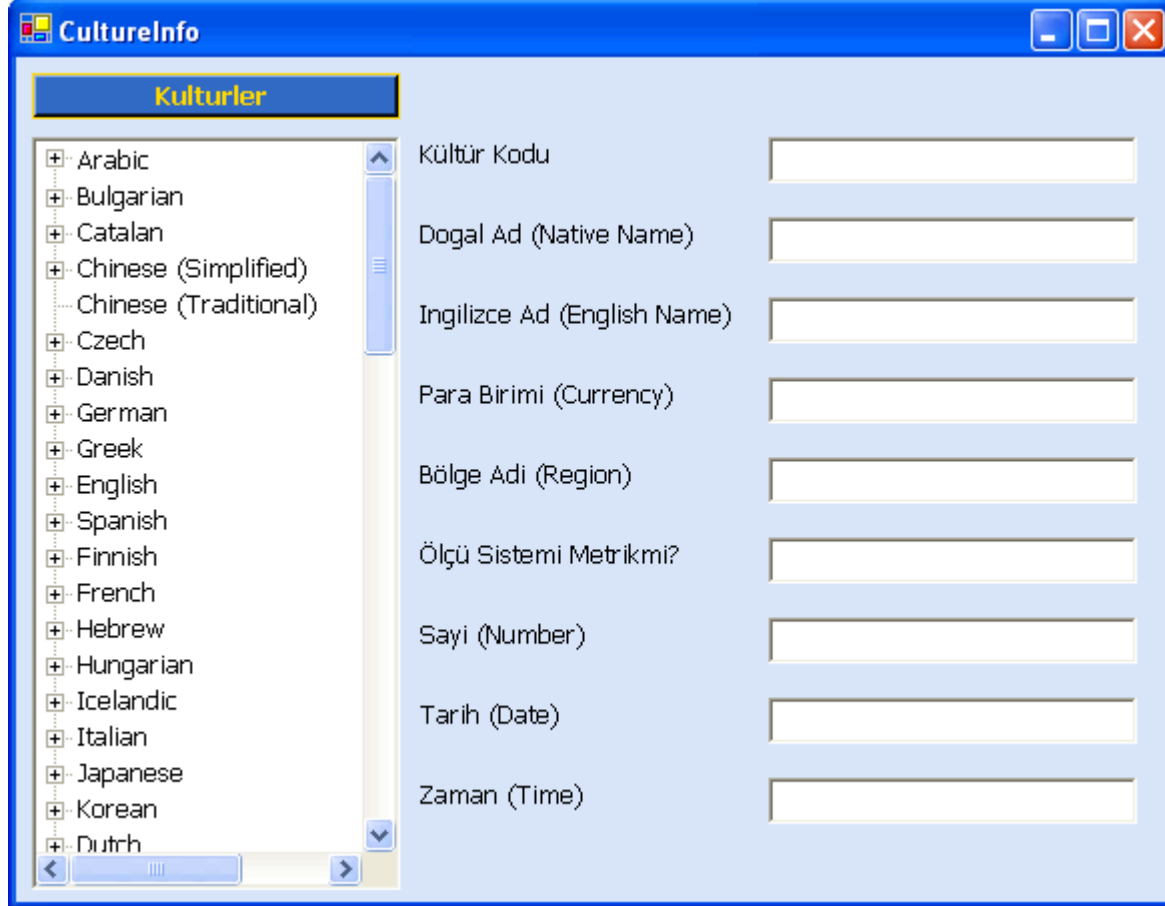
        txtTime.Text=DateTime.Now.ToString("T");
    }
}
}

```

TreeView kontrolünde her hangibir öge seçildiğinde bu ögeye ait bilgiler, textBox kontrollerine doldurulur. Eğer seçilen öge doğal bir kültür değilse, yani belirleyici kültür ise, bu kültürün bulunduğu ülke veya bölgeye ait bilgilere ulaşmak için RegionInfo sınıfına ait bir nesne kullanılır. Bu nesnenin belirleyici bir kültüre ait olarak oluşturulması sırasında, belirleyici kültürü temsil eden bir değer kullanılır. Bu değer, CultureInfo sınıfının LCID özelliği ile elde edilen benzersiz bir belirleyicidir. RegionInfo nesnesi yardımıyla, belirleyici kültürün bulunduğu bölgeye has para birimi, bu bölgede metrik sistemin kullanılıp kullanılmadığına dair belirleyici bilgiler elde edilir. Gelelim işin en önemli kısmına. Yani uygulamanın seçilen kültüre göre yerleştirildiği (Localization) kısma.

```
Thread.CurrentThread.CurrentCulture=guncelKultur;
```

Bu kod satırı ile, çalışan proses için kültür değeri, seçilen kültüre göre ayarlanmıştır. Dolayısıyla bu kod satırını izleyen satırlardaki sayısal değer, tarih ve zaman formatları, seçilen kültüre göre ekrana gelecektir. İşte uygulamamız basit bir şekilde yerleştirme işlemini gerçekleştirmiştir. Şimdi uygulamamızı çalıştıralım ve sonuçları gözlemleyelim. Programı ilk çalıştırdığımızda ve Kulturler başlıklı butona tıkladığımızda aşağıdaki ekran görüntüsünü elde ederiz.



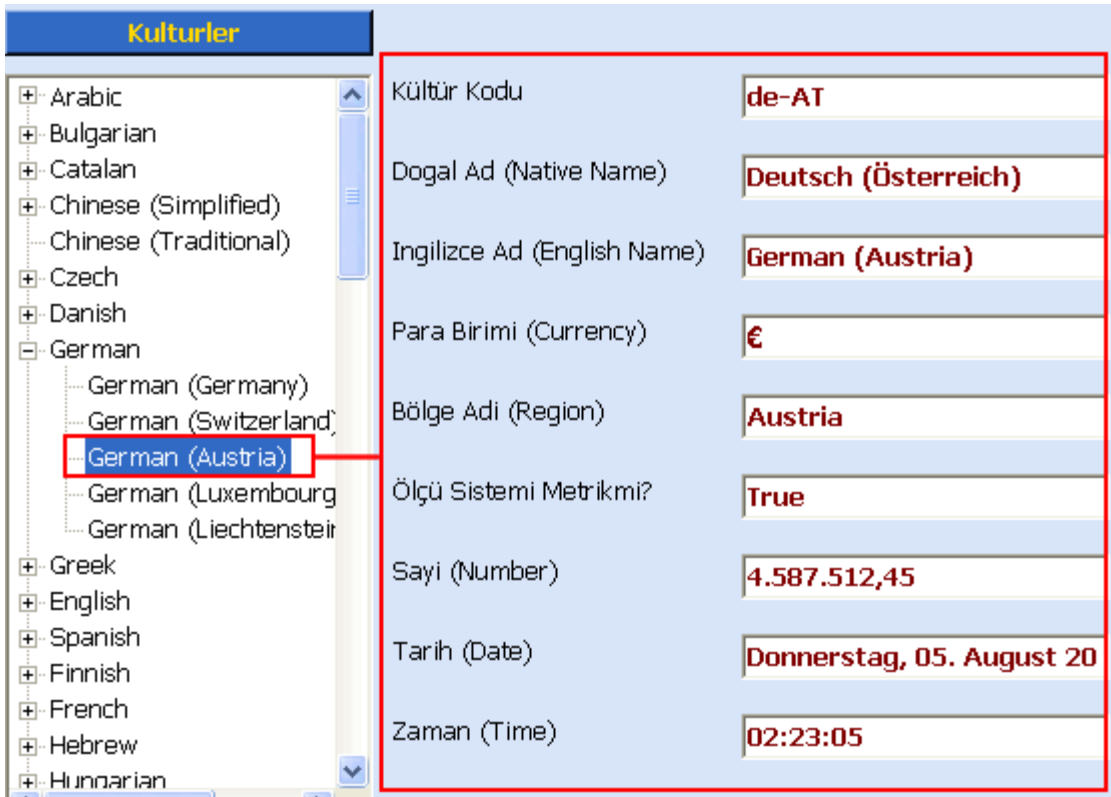
Şekil 3. Tüm Kültürler.

Örneğin German doğal kültürünü seçelim. Bu durumda Almanca dilini konuşan alt kültürler görünecek ve Almanca doğal kültürüne ait kültür kodu, doğal adı ve ingilizce ad bilgileri elde edilecektir.



Şekil 4. Almanca (German) Doğal Kültürü.

Şimdi belirleyici kültürlerden birini seçelim. Örneğin Avusturya alt kültürünü. Bu durumda bu kültüre ait yerel bilgiler ekrana gelecek ve uygulama bu noktadan sonra, Avusturya'ya ait yerel bilgilere göre sayıları formatlayacak, tarihleri gösterecek vb... işlemleri gerçekleştirecektir.



Şekil 5. Avusturya Kültürüne ait özellikler.

Şimdi daha uç bir örnek gösterelim. Örneğin uygulamamızın Suudi Arabistan kültürüne göre yerelleştirilmek istendiğini farzedelim. Bu durumda ilk akla gelen arap takviminin farklılığıdır.

Şekil 6. Arabistanda durum dahada karışık.

Böylece bir uygulamanın, belirli bir kültüre göre yerelleştirilmesini görmüş olduk. Diğer taraftan, uygulamamızın farklı dillere göre destek vermesinde isteyebiliriz. İşte bu noktada devreye Resource Files (Kaynak Dosyalar) girmektedir. Bu konuyu ise bir sonraki makalemizde incelemeye çalışacağız. Hepinize mutlu günler dilerim.

Localization (Yerelleştirme) 2 - Dil Desteği

Değerli Okurlarım, Merhabalar.

Bir önceki makalemizde hatırlayacağınız gibi, .net ile geliştirilen uygulamaların belirli kültürler için nasıl yerelleştirilebileceğini incelemeye başlamıştık. İlk bölümde, belirli bir kültürün daha çok sayısal, tarihsel ve sıralama formatları üzerinde durduk. Bu bölümde ise, yerelleştirmede daha da önemli olan bir konuya, uygulamaların farklı dillere göre destek vermesine değineceğiz.

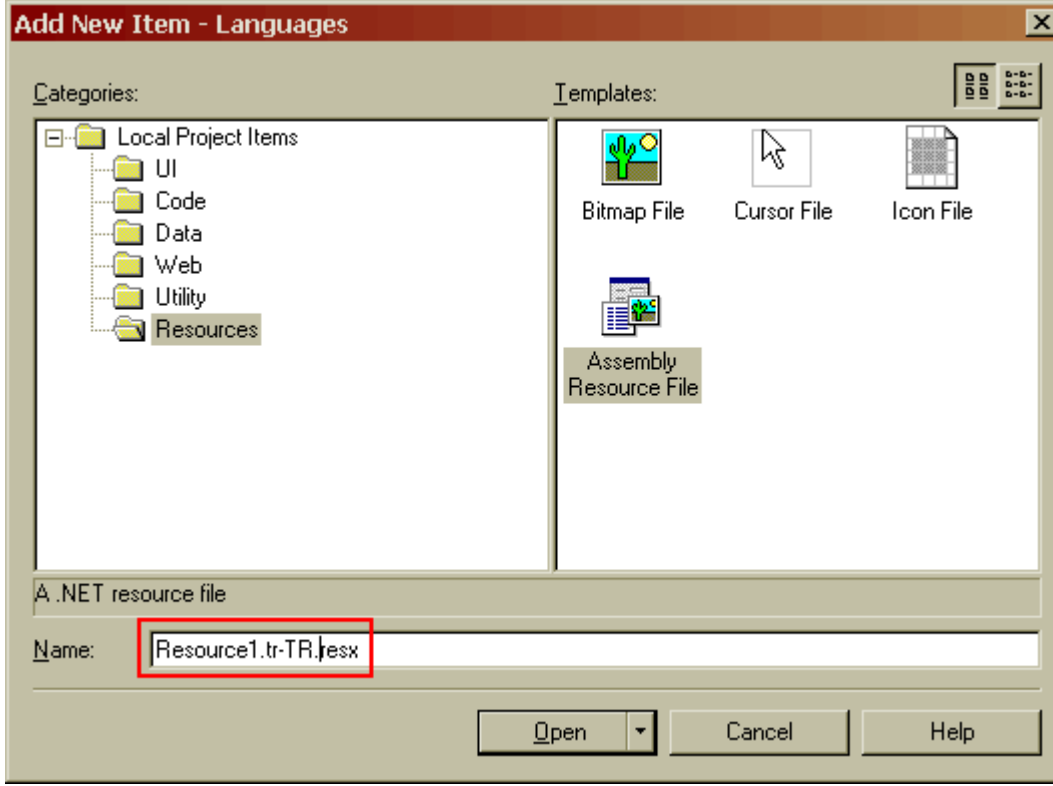
Bir .net uygulamasının farklı dillere destek vermesindeki anahtar nokta, Resource dosyalarının etkin bir şekilde kullanılmasında yatmaktadır. Örneğin, uygulamamızın arayüzünün Türkçe olarak oluşturulduğunu düşünelim. Bu uygulamaya ait arayüzdeki metinleri, başka bir dilde sunabilmek için, ilgili kelimelerin diğer dildeki karşılıklarının bir şekilde assembly tarafından bilinmesi

gerekir. Sadece bilinmesi elbette yeterli değildir. Ayrıca, çalışan proses için, arayüz kültüründe istenen dile ayarlanması gerekmektedir.

Şimdi dilerseniz bu işlemlerin nasıl gerçekleştirilebildiğini basit bir uygulama üzerinde anlamaya çalışalım. Uygulamamız aşağıdaki form görüntüsüne sahip bir windows uygulaması olacak. Şu anda form üzerindeki tüm metinler Türkçe. Biz uygulamamızı, İngilizce, Fransızca ve Almanca dillerine destek verecek hale getireceğiz.

Şekil 1. Uygulama Form Tasarımımız.

İlk olarak Resource dosyalarını oluşturacağız. Bu dosyaları, birer sözlük olarak düşünebiliriz. Resource dosyaları XML tabanlı dosyalar olup, temelde verileri anahtar yada isim (key - name) - değer (value) çiftleri şeklinde tutmaktadırlar. Bu açıdan bakıldıklarında bunları Hash Tablolarına benzetebiliriz. Dolayısıyla, uygulamamıza dil desteği sağlamak istiyorsak, ortak isimler (names) belirleyip, ilgili dil için uygun değerleri atayacağımız dosyalar oluşturmamız gerekecektir. Yani Türkçe için bir tane, İngilizce için bir tane, Almanca için bir tane ve Fransızca için bir tane. Bir Resource dosyasını Vs.Net ortamında uygulamaya eklemek için projeye sağ tıklayıp Add New Item iletişim kutusuna girmemiz ve Assembly Resource File dosya tipini seçmemiz yeterlidir. Burada önemli olan Resource dosyasının isimlendiriliş şeklidir.



Şekil 2. Resource Dosyasının Eklenmesi.

Dikkat edecek olursanız dosyamızı isimlendirirken tr-TR takısını kullandık. Bu Türkçe dilini konuşan Ülkemizi temsil eden belirleyici bir kültür kodudur. Dosyanın bu şekilde isimlendirilmesinin sebebi, çalışma zamanında, seçilen kültüre göre hangi Resource dosyasındaki kelimelerin yükleneceğinin belirlenebilmesidir. Bunu kodlarımızı yazdığımızda daha net bir şekilde açıklamaya çalışacağım. Şimdi dosyamızın name ve value değerlerini girelim.

Data for data					
	name	value	comment	type	mimetype
	Dil	Dil	(null)	(null)	(null)
	Tarih	Tarih	(null)	(null)	(null)
	Saat	Saat	(null)	(null)	(null)
	Parasal	Parasal	(null)	(null)	(null)
	Göster	Göster	(null)	(null)	(null)
▶	Türkçe Dil	Türkçe Dil	(null)	(null)	(null)

Şekil 3. Resource1.tr-TR.resx dosyasının içeriği.

Artık tek yapmamız gereken, diğer diller içinde gerekli olan resx dosyalarını oluşturmak. Bu dosyalarda, name yada key değerlerimiz Türkçe olacak. Ancak karşılıkları olan kelimeler value değeri olarak belirlenecek. Örneğin, Fransızca için aşağıdaki bilgiler girilecek.

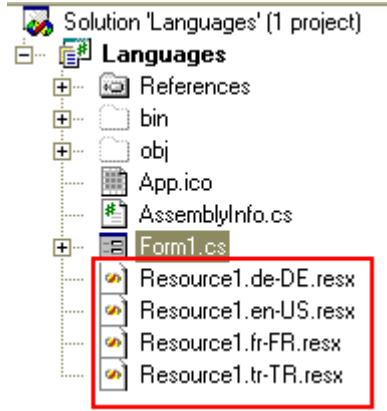
orm1.cs | Resource1.en-US.resx | Resource1.tr-TR.resx | **Resource1.fr-FR.resx** | Resource1.de-DE.resx

Data:

Data for data					
	name	value	comment	type	mimetype
►	Dil	langue	(null)	(null)	(null)
	Tarih	date	(null)	(null)	(null)
	Saat	temps	(null)	(null)	(null)
	Parasal	monnaie	(null)	(null)	(null)
	Göster	démonstratio	(null)	(null)	(null)
	Türkçe Dil	français lang	(null)	(null)	(null)

Şekil 4. fr-Fr kültürü için dil bilgilerinin Resource dosyasına girilmesi.

Sonuç olarak, projemizde aşağıdaki Resource dosyalarının oluşturulması gerekmektedir. Bu sayede, uygulamamız dünyanın neresinde olursa olsun, kaynak dosyalarda belirtilen dillere ve kültürlere destek verebilecektir. Elbette, her resx dosyasında name alanlarına yazılan Türkçe kelimelerin diğer dildeki karşılıkları value alanına girilecektir.



Şekil 5. Resx dosyalarımız.

Şimdi uygulama kodlarımızı yazmaya başlayalım. Uygulamamız için, yerelleştirme söz konusu olduğundan, System.Globalization isim alanını ve proses için kültürel özellikleri ve dil özelliklerini belirleyeceğimizden System.Threading isim alanlarını ilk olarak eklemeliyiz. Bunların yanında, assembly içindeki Resource dosyalarını yönetebilmemiz için, ResourceManager sınıfından bir nesneye ihtiyacımız olacaktır. Bu nesne içinde, System.Resources isim alanını uygulamamıza eklemeliyiz. İşte kodlarımız,

```
private void Doldur()
{
    this.textBox1.Text=DateTime.Today.ToLongDateString();
    this.textBox2.Text=DateTime.Now.ToLongTimeString();
    double sayi=121345.4565;
    this.textBox3.Text=sayi.ToString();
}
private void Form1_Load(object sender, System.EventArgs e)
```

```

{
    Doldur();
}

private void button1_Click(object sender, System.EventArgs e)
{
    if(comboBox1.SelectedItem.ToString()=="USA")
    {
        Thread.CurrentThread.CurrentUICulture=new CultureInfo("en-US");
        Thread.CurrentThread.CurrentCulture=new CultureInfo("en-US");
    }
    else if(comboBox1.SelectedItem.ToString()=="Türkiye")
    {
        Thread.CurrentThread.CurrentUICulture=new CultureInfo("tr-TR");
        Thread.CurrentThread.CurrentCulture=new CultureInfo("tr-TR");
    }
    else if(comboBox1.SelectedItem.ToString()=="Français")
    {
        Thread.CurrentThread.CurrentUICulture=new CultureInfo("fr-FR");
        Thread.CurrentThread.CurrentCulture=new CultureInfo("fr-FR");
    }
    else if(comboBox1.SelectedItem.ToString()=="Deutschland")
    {
        Thread.CurrentThread.CurrentUICulture=new CultureInfo("de-DE");
        Thread.CurrentThread.CurrentCulture=new CultureInfo("de-DE");
    }

    ResourceManager resM=new ResourceManager("Languages.Resource1",Type.GetType("Languages.Form1").Assembly);
    this.Text=resM.GetString("Türkçe Dil");
    this.lblDil.Text=resM.GetString("Dil");
    this.lblParasal.Text=resM.GetString("Parasal");
    this.lblSaat.Text=resM.GetString("Saat");
    this.lblTarih.Text=resM.GetString("Tarih");
    this.button1.Text=resM.GetString("Göster");
    Doldur();
}

```

Kullanıcı comboBox1 kontrolünden bir dil seçtiğinde, öncelikle uygulamanın arayüzünün hangi dili kullanacağı CurrentUICulture özelliği ile belirlenmektedir. İşte bu noktada, resource dosyalarımız isimlendirilirken neden kültür kodlarının kullanıldığı dahada belirginleşmektedir. Nitekim, bu özellik, CultureInfo türünden bir nesne alır. Bu nesne belirli bir kültür için oluşturulmaktadır. Bu

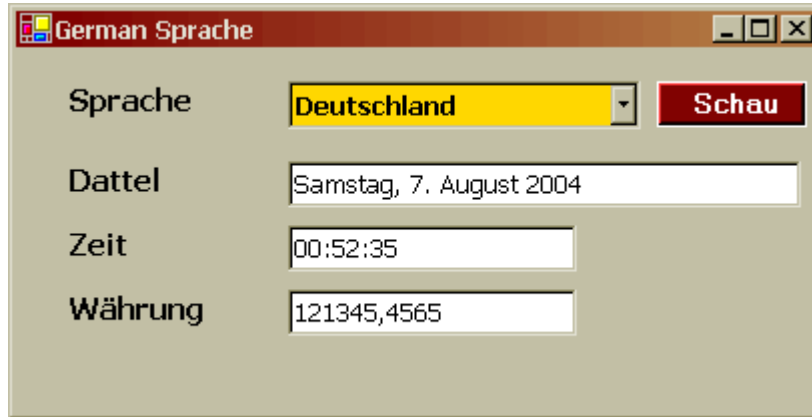
kültürün kodu, uygulama tarafından Resource dosyalarının isimlerinde aranır. Yani kullanıcı USA' i seçtiğinde, uygulama arayüzü için en-US kültür kodu belirlenir. Dolayısıyla, ResourceManager nesnesi, bu kültür kodunu içeren Resource dosyasını kullanmaya başlayacaktır.

If koşullarında, sonraki hamlede, güncel proses' deki sayısal,tarihsel vb... formatlar için gerekli kültür kodu, CurrentCulture özelliğine, ilgili kültür için bir CultureInfo nesnesi atanarak gerçekleştirilir. Bizim için önemli olan bir diğer noktada, ResourceManager nesnesinin oluşturuluş şeklidir.

```
ResourceManager resM=new ResourceManager("Languages.Resource1",Type.GetType("Languages.Form1").Assembly);
```

Burada ilk parametre, Resource dosyalarının ana adını işaret etmektedir. Ana adımız örneğin Resource1.tr-TR.resx dosyasının baz aldığımızda, belirleyici kültür koduna kadar olan kısımdaki dosya adıdır. İkinci parametre ise, Reflection özelliklerini kullanır ve güncel assembly' ın tipini alır. Artık elimizde, resource' ları yönetebileceğimiz bir nesne vardır. Tek yapmamız gereken, uygulama arayüzündeki text' lere, ResourceManager nesnesinin GetString metodu ile, name (key) alanlarının karşılığı olan değerlerin (value) atanmasıdır. Bunun için, ResourceManager sınıfının GetString metodunu kullandık.

Şimdi akla şu soru gelebilir. ResourceManager hangi resx dosyasını, dolayısıyla hangi dili kullanacağını nereden bilecek. İşte bunu sağlayan, az önce bahsettiğimiz CurrentThread sınıfının CurrentUICulture özelliğidir. Bu özelliğe atadığımız belirleyici kod burada devreye girerek, ResourceManager' a hangi resx dosyasını kullanması gerektiğini söylemektedir. Bu açıklamalardan sonra dilerseniz, uygulamamızı derleyip çalıştıralım. Her hangibir dili seçtiğimizde, hem formatların hemde arayüzdeki metinlerin ilgili dile göre değiştiğini görürüz. İşte arayüzdeki metinlerin seçilen dildeki karşılıkları, oluşturduğumuz resx dosyalarından, ResourceManager sınıfının GetString metodu ile alınmaktadır.



Şekil 6. Almanca.

The screenshot shows a window titled 'English Language'. It contains four rows of settings: 'Language' with a dropdown menu set to 'USA' and a 'Show' button; 'Date' with a text box containing 'Saturday, August 07, 2004'; 'Time' with a text box containing '12:52:11 AM'; and 'Currency' with a text box containing '121345.4565'.

Şekil 7. İngilizce.

The screenshot shows a window titled 'français langue'. It contains four rows of settings: 'langue' with a dropdown menu set to 'Français' and a 'démonstr' button; 'date' with a text box containing 'samedi 7 août 2004'; 'temps' with a text box containing '00:52:59'; and 'monnaie' with a text box containing '121345,4565'.

Şekil 8. Fransızca

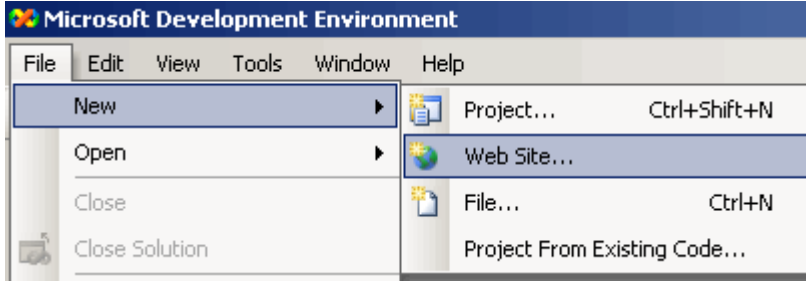
Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Asp.Net 2.0 ile Cross-Page Posting

Değerli Okurlarım Merhabalar,

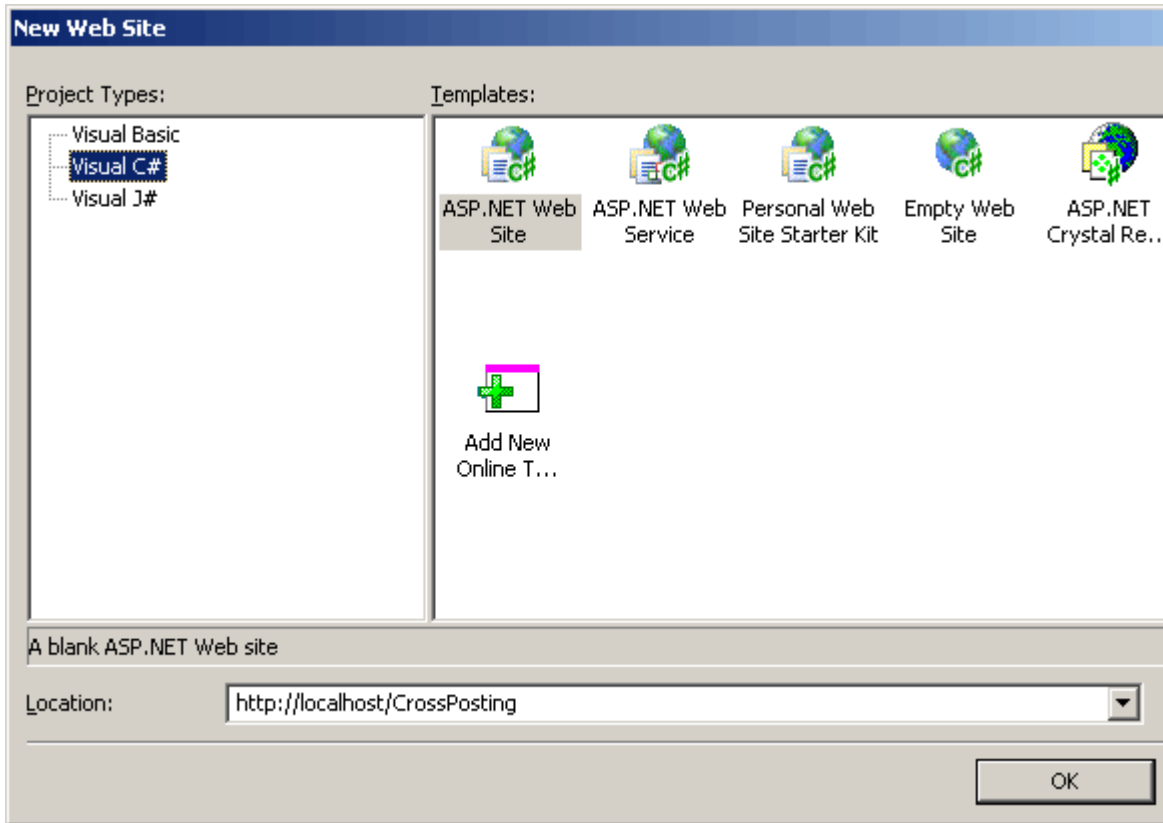
Bu makalemizde, Asp.Net 2.0 (Asp.Net Whidbey) ile, başka sayfalara veri postalama işlemlerinin nasıl ele alındığını ve uygulandığını incelemeye çalışacağız. Bildiğiniz gibi, Asp.Net 1.0 / 1.1 ile gelen yeniliklerden en önemlisi, sayfaların kendi kendilerine form verilerini postalayabilme kabiliyetleridir. Öyleki, Asp.Net ile geliştirilen web sayfaları aslında birer sınıf nesnesi olduklarından, sayfa üzerindeki form kontrollerine ve değerlerine kolayca erişilebilmektedir. Ancak bazı zamanlarda, sayfalarımızda yer alan form verilerini başka sayfalara göndermek isteyebiliriz. İşte Cross-Page Posting olarak adlandırılan bu işlemlerin, Asp.Net 2.0 ile gerçekleştirilmesi hem daha kolay hemde daha etkili hale getirilmiştir.

Örneklerimizi geliştirdiğimizde bu konuyu çok daha iyi anlayabileceğinize inanıyorum. Hiç vakit kaybetmeden örneğimizi geliştirmeye başlayalım. Bu örnek uygulamamızı, Visual Studio.Net' in 2005 sürümünün Beta versiyonu ile geliştireceğiz. Örneğimizde, basit olarak iki web sayfası kullanacağız. Öncelikle yerel sunucumuzda bir web sitesi oluşturalım. Bunun için, File menüsünden, New alt menüsünü ve buradan da Web Site kısmını seçiyoruz.



Şekil 1. Yeni bir Web Site açıyoruz.

Ardından, karşımıza çıkacak dialog penceresinden, Asp.Net Web Site' ı seçelim ve localhost altında CrossPosting isimli sanal klasörümüzün adını girelim.



Şekil 2. Web Site'ımızı oluşturuyoruz.

Bu işlemin ardından, Visual Studio.Net standart olarak default.aspx isimli sayfamızı oluşturur. Bu sayfamızı aşağıdaki şekilde olduğu gibi oluşturalım.

Şekil 3. Form tasarımı.

Sayfamızın kodlanmasına geçmeden önce, aspx sayfamızın içeriğine bir göz atalım. Nitekim burada önemli olan btnDiger ID değerine sahip button kontrolümüzün takıları arasındaki içeriktir. Biz burada ufak bir değişiklik yapacağız. Önce kodlarımızın başlangıç haline bakalım.

```
<%@ Page Language="C#" CompileWith="Default.aspx.cs" ClassName="Default_aspx" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
<head runat="server">
```

```
<title>Untitled Page</title>
```

```
</head>
```

```
<body>
```

```
<form id="form1" runat="server">
```

```
<asp:Label ID="Label1" Runat="server" Text="Rumuzunuz" Width="86px" Height="19px"
```

```
Font-Bold="True"></asp:Label><span style="font-size: 10pt; font-family: Verdana"> </span>
```

```
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox><br />
```

```
<span style="font-size: 10pt; font-family: Verdana"></span>
```

```
<asp:Label ID="Label2" Runat="server" Text="Yaş Aralığınız" Font-Bold="True"></asp:Label><span  
style="font-size: 10pt; font-family: Verdana"> </span>
```

```
<asp:DropDownList ID="DropDownList1" Runat="server" Width="71px" Height="22px">
```

```
<asp:ListItem>20-25</asp:ListItem>
```

```
<asp:ListItem>25-30</asp:ListItem>
```

```
<asp:ListItem>30-35</asp:ListItem>
```

```
</asp:DropDownList><span style="font-size: 10pt; font-family: Verdana">
```

```
<br />
```

```
<br />
```

[illegible]

Burada btnDiger, ID değerine sahip button kontrolümüzün olduğu satırı aşağıdaki gibi değiştirelim.

```
<asp:Button ID="btnDiger" PostBackUrl="Diger.aspx" Runat="server" Text="Diger" Width="58px" Height="24px" />
```

İşte Asp.Net 2.0 ile gelen ilk yeniliklerden birisi. PostBackUrl özelliği. Bu Url, button kontrolüne basıldığında, Form üzerindeki verilerin belirtilen url' deki sayfaya gönderilmesini sağlamaktadır. İşleyiş şekli son derece etkilidir. Şimdi Solution Explorer' dan Add New Item ile, Diger.aspx isimli sayfamızda uygulamamıza ekleyelim. Ardından, default.aspx sayfamızın Code-Behind kodlarını da aşağıdaki gibi geliştirelim.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class Default_aspx
{
    void btnBurası_Click(object sender, EventArgs e)
    {
```

```

        lblKim.Text = TextBox1.Text + DropDownList1.SelectedItem.Text;
    }
}

```

Burada yaptığımız değişik bir olay yok. Sadece, TextBox1 kontrolüne girilen ve DropDownList1 kontrolünde seçilen değerleri, Label kontrolüne yazdırdık. İşte bu, Asp.Net 1.0/1.1 sürümünden bildiğimiz, sayfanın kendi kendine verileri postalaması işlemidir. Asp.Net 2.0 ile gelen yeni teknikleri ise, Diger.aspx sayfasında kullanacağız. Bunun için Diger.aspx sayfamıza bir Label kontrolü yerleştirelim ve ardından, sayfamıza ait kodları aşağıdaki gibi düzenleyelim.

```

using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

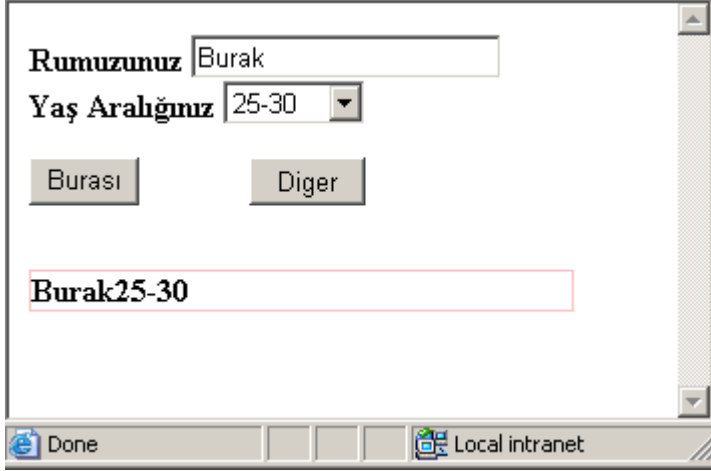
public partial class Diger.aspx
{
    void Page_Load(object sender, EventArgs e)
    {
        TextBox tbRumuz = (TextBox)PreviousPage.FindControl("TextBox1");
        DropDownList lstYas = (DropDownList)PreviousPage.FindControl("DropDownList1");

        lblDurum.Text = tbRumuz.Text + lstYas.SelectedItem.Text;
    }
}

```

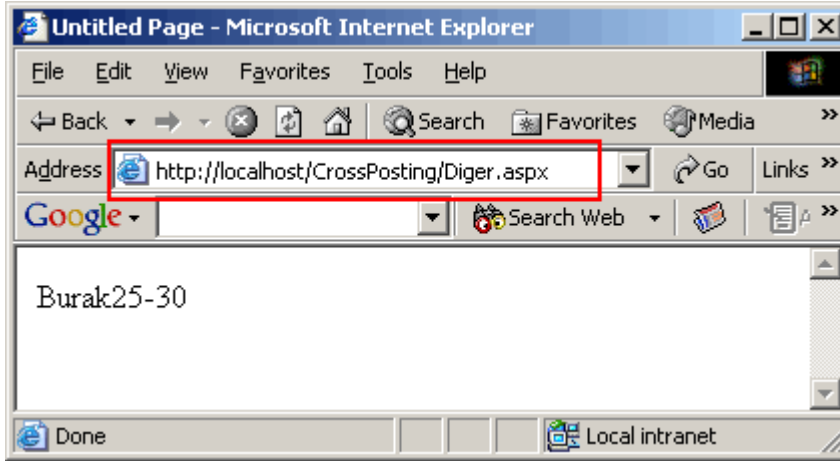
Dikkat edecek olursanız burada son derece güçlü bir teknik kullanılmaktadır. Bir adet TextBox ve bir adette DropDownList kontrolü tanımlanmış ve oluşturulmuştur. TextBox kontrolümüzü oluştururken, PreviousPage sınıfının FindControl metodu ile, bu sayfaya Post işlemi ile form verisi gönderen sayfadaki TextBox1 kontrolü bulunmaktadır. Bu işlem, bulunan kontrolün TextBox kontrolü olarak cast edilmesi ile tamamlanır. Dolayısıyla, Diger.aspx sayfamızın Page_Load olay metodunda, elimizde bir adet TextBox kontrolü olacaktır. Bu TextBox kontrolü aslında, default.aspx sayfasından gelen kontroldür. Böylece, default.aspx sayfasındaki TextBox1 kontrolünün değerine ve özelliklerine bu sınıf içerisinde (Diger.aspx sayfasından) kolayca erişebiliriz. Aynı teknik, DropDownList kontrolümüz içinde geçerlidir. İşte Asp.Net 2.0, başka sayfalara form verisi gönderme işlemlerinde böylesine güçlü bir teknik ile gelmektedir.

Uygulamamızı çalıştırdığımızda ve Burası başlıklı butona tıkladığımızda basitçe sayfa kendi kendisine postalama işlemini uygular. (Postback).



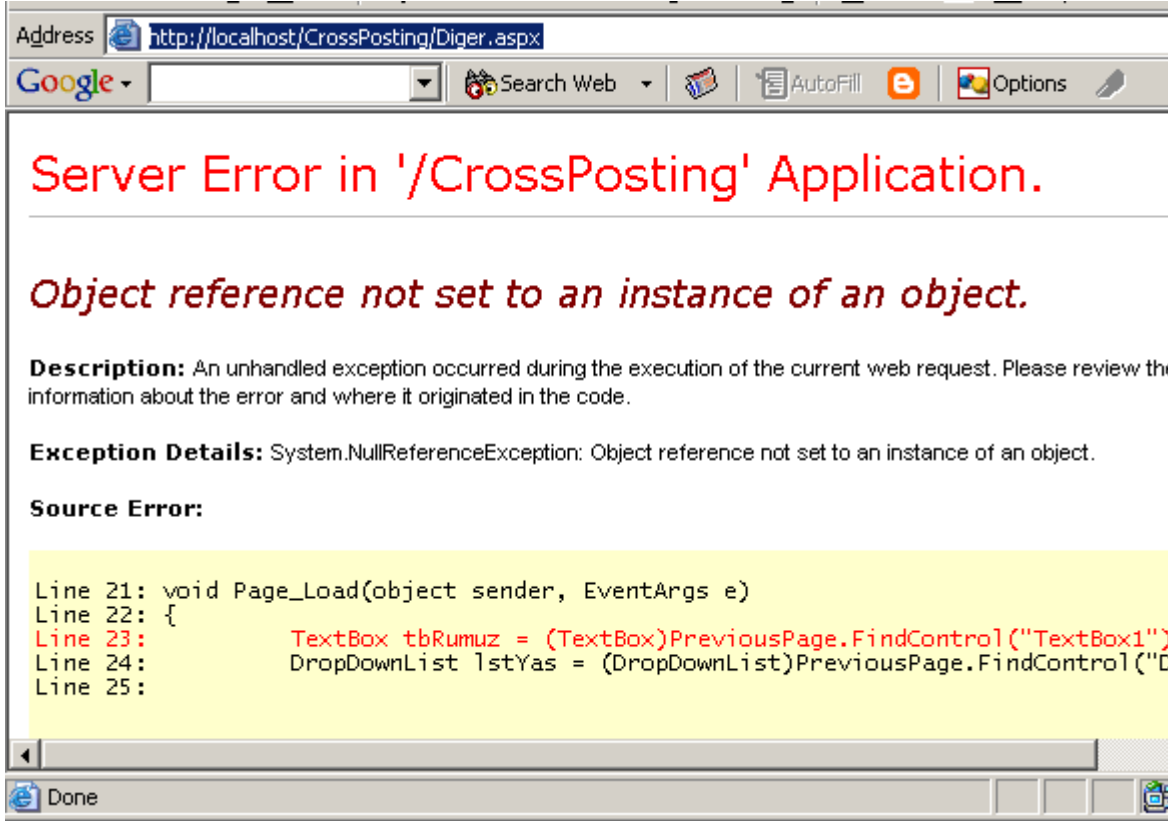
Şekil 4. Postback.

Diger başlıklı button kontrolüne bastığımızda ise, Diger.aspx isimli sayfaya gidilir. Bu sayfanın Page_Load olay metodu çalışır ve burada ilgili TextBox ve DropDownList kontrolleri, default.aspx' ten gelen kontroller baz alınarak oluşturulur. Sonuç olarak, default.aspx sayfasındaki kontroller üzerinde yer alan veriler kolayca elde edilir ve bu sayadaki Label kontrolüne içerikleri yazdırılır.



Şekil 5. Cross-Page Posting

Cross-Page Posting işleminde elbette sorun yaratabilecek durumlarda söz konusudur. Bunların en önemlisi, bir kullanıcının, default.aspx çalıştırılmadan önce diger.aspx sayfasına ulaşmaya çalışmasıdır. Böyle bir durumda, default.aspx daha önceden çalıştırılmadığı için, buradaki kontroller oluşturulmamış olacaktır. Dolayısıyla, diger.aspx sayfasında, PreviousPage sınıfına ait metodlar, var olmayan kontrolleri bulmaya çalışacak ve referanslar oluşturulamayacaktır. Kullanıcıların böyle bir girişimde bulunması sonucunda aşağıdaki gibi bir hata mesajı ile karşılaşılır.



Şekil 6. Oluşan Hata.

Çözüm gayet basittir. Bu sayfa çalıştırıldığında, bu sayfaya başka bir sayfadan form bilgisinin gönderilip gönderilmediği öğrenilmelidir. Bunun için, Page sınıfının IsCrossPagePostBack isimli özelliğinden yararlanılır. Bu özellik, aslında IsPostBack özelliği gibi çalışır. Sadece, başka bir sayfanın mevcut sayfaya veri postalayıp postalamadığını kontrol eden ve boolean değer döndüren bir yapıdadır. Prototipi aşağıdaki gibidir.

```
public bool IsCrossPagePostBack {get;}
```

Dolayısıyla, diger.aspx sayfamızın arka kodlarını aşağıdaki gibi değiştirmemiz sorunun giderilmesini sağlayacaktır.

```
public partial class Diger_aspx
{
    void Page_Load(object sender, EventArgs e)
    {
        if (Page.IsCrossPagePostBack)
        {
            TextBox tbRumuz = (TextBox)PreviousPage.FindControl("TextBox1");
            DropDownList lstYas = (DropDownList)PreviousPage.FindControl("DropDownList1");

            lblDurum.Text = tbRumuz.Text + lstYas.SelectedItem.Text;
        }
    }
}
```

```

    }
    else
    {
        Response.Redirect("default.aspx");
    }
}
}

```

Böylece, eğer kullanıcılar direkt olarak `diger.aspx` sayfasını çalıştırlırsa, önceden gelen bir Cross-Page Posting işlemi olmadığından, else bloğundaki kod satırı çalışacak ve `default.aspx` sayfasına gidilecektir.

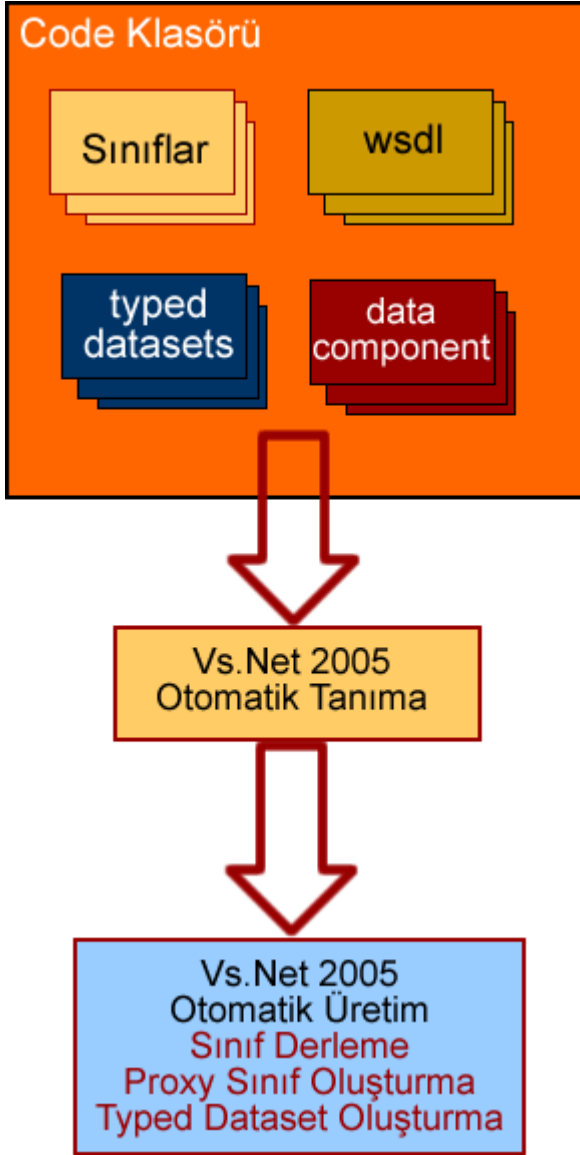
Bu makalemizde, Asp.Net 2.0 ile gelen yeniliklerden birisine değinmeye çalıştık. Elbette şu an için, Asp.Net 2.0 henüz beta aşamasında. Yani yazılan kodlar ve kullanılan teknikler değişebilir hatta bazen kaynakların aksine çalışmayabilir. Bunlardan da haberdar oldukça sizleri bilgilendirmeye çalışacağız. Böylece geldik bir makalemizin daha sonuna. İlerleyen makalelerimizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Asp.Net 2.0 ve Code Klasörü

Değerli Okurlarım Merhabalar,

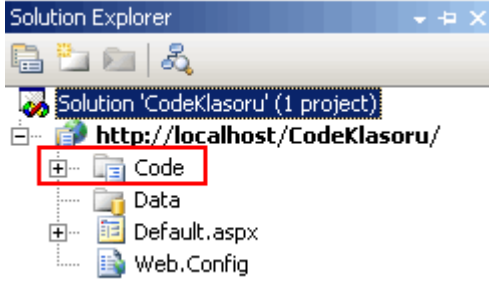
Bu makalemizde, Asp.Net 2.0 ile gelen tanımlanmış klasörlerden (defined folders), Code klasörünün nasıl kullanıldığını incelemeye çalışacağız. Asp.Net 1.0/1.1 ile uygulama geliştirirken, solution içindeki herşey bir dll içinde (assembly) toplanır. Asp.Net 2.0 ise, dosya tabanlı (file-based) yaklaşım adı verilen yeni bir teknik kullanır. Bu tekniğe göre, solution, dosyalar ve klasörler sisteminden oluşmaktadır. Bu sistemin faydası, otomatik derleme özelliğine sahip olmasıdır. Yani, solution içerisine herhangi bir dosya eklenmesi halinde (örneğin bir sınıf), Visual Studio.Net 2005 bu dosyayı otomatik olarak derler ve solution' ın her zaman dinamik olarak güncel kalmasını sağlar. Bir başka deyişle yeni eklenen dosya için, solution' ın baştan derlenmesi gerekmez. Bu yüzden, Asp.Net 2.0 ile uygulama geliştirirken ilk dikkati çeken unsur, oluşturulan sanal klasör altında eskiden olduğu gibi bin klasörünün ve bir dll dosyasının olmayışdır.

Asp.Net 2.0 ile geliştirilen bu yeni teknikte, çeşitli önceden tanımlanmış (predefined folders) özel klasör seçenekleri mevcuttur. Bunlardan birisi de Code klasörüdür. Code klasörü, sınıf dosyalarımız, web servisleri için kullanılan wsdl dosyalarımız, türlendirilmiş dataset' ler (typed datasets) için kullanılan xml şemaları ve Data Component' ler tipindeki dosyaları barındırılabilir. Code klasörü içerisine konulan bu dosyalar, Visual Studio.Net 2005 ortamında otomatik olarak tanınır. Ayrıca, Visual Studio.Net 2005, bu dosyaları kullanarak, sınıflara derler, proxy sınıflarını veya türlendirilmiş veri sınıflarını oluşturur. Bu yapıyı aşağıdaki şekil ile daha kolay anlayabiliriz.



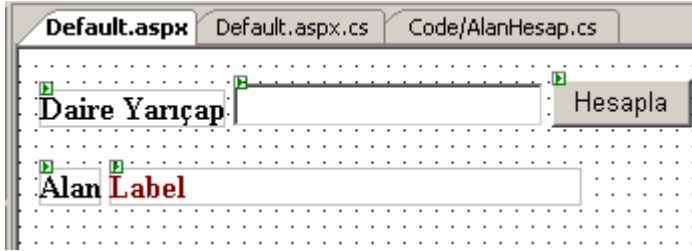
Şekil 1. Genel yapı.

Dilerseniz konuyu daha iyi anlayabilmek amacıyla basit bir örnek geliştirelim. Bunun için Visual Studio.Net 2005' de yeni bir Web Site açıyoruz. Daha sonra Solution Explorer' da solution' ımız üzerine sağ tıklayıyor ve New Folder seçeneğine basarak, yeni bir klasör oluşturuyoruz. Klasörümüze Code ismini verdiğimizde, şeklinin normal klasörlerden biraz daha farklı olduğunu hemen farkedebiliriz. Nitekim Code klasörünün Solution için özel bir anlamı vardır.



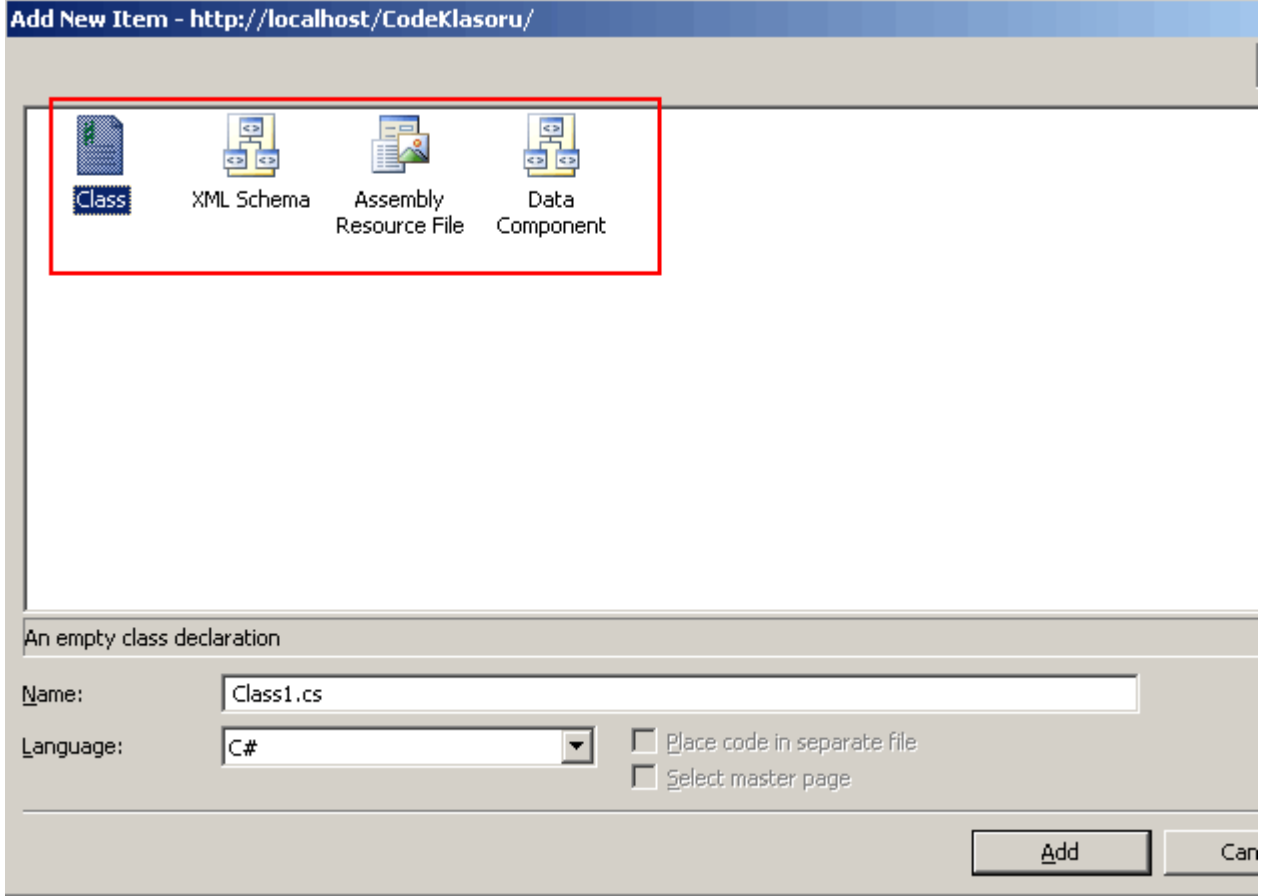
Şekil 2. Code klasörünün eklenmesi.

Şimdi default.aspx form' unuda aşağıdaki gibi oluşturalım. Bu web sayfasında basit olarak, yarıçapı verilen bir dairenin alanı hesap edilecek. Bu hesaplama işlemini yapan metodumuzu barındıracak bir sınıfımız olacak ve bu sınıfımız, Code klasörü içerisinde yer alacak.



Şekil 3. Form tasarımı.

Şimdi, Code klasöründe sağ tıklayalım ve Add New Item' i seçelim. Karşımıza aşağıdaki dialog penceresi çıkacaktır.



Şekil 4. Code klasörü için yeni bir öğe eklemek.

Bu dialog penceresinde, Code klasörüne ekleyebileceğimiz dosya tipleri yer almaktadır. Biz class tipini seçeceğiz. AlanHesap.cs dosyamızın kodları aşağıda görüldüğü gibidir.

```
using System;
```

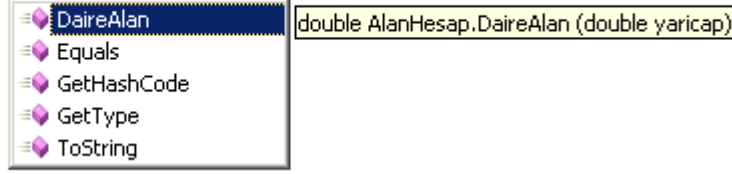
```
public class AlanHesap
{
    public AlanHesap()
    {

    }

    public double DaireAlan(double yaricap)
    {
        return 3.14 * (yaricap * yaricap);
    }
}
```

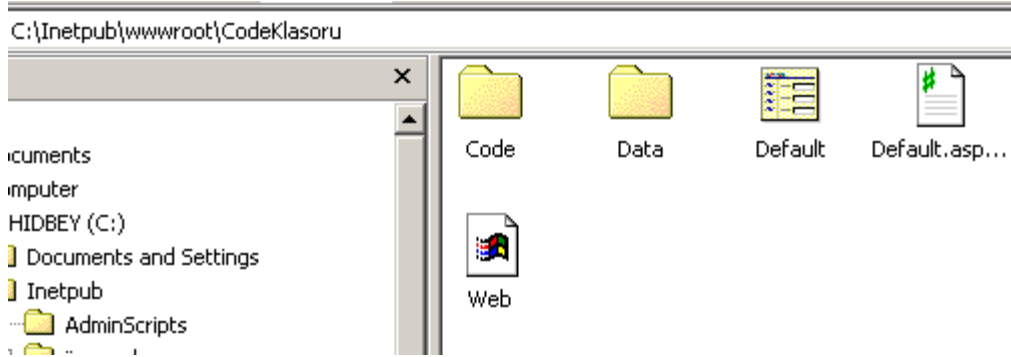
Bu noktadan sonra, solution' ımızı hiç derlemeden, AlanHesap sınıfımızı kullanabilir, bu sınıftan nesne örnekleri yaratabilir daha da önemlisi intelli-sense özelliğinden derhal faydalanabiliriz.

```
AlanHesap ah = new AlanHesap();
lblSonuc.Text = ah.
```



Şekil 5. Intelli-Sense Özelliği.

Bu noktadan sonra, uygulamamızı oluşturduğumuz klasöre bakarsak aşağıdaki yapıda olduğunu farkederiz. Dikkat edecek olursanız Asp.Net 1.0/1.1' deki gibi kalabalık bir topluluk yoktur. En önemlisi Bin klasörünü veya tüm uygulamanın tiplerine ait manifesto bilgilerini ve kodları barındıran bir dll görememekteyiz.



Şekil 6. Klasör Azlığı.

Code klasörü için dikkat edilecek noktalardan birisi, buradaki dosyaların tamamının single assembly olarak ele alınmalarıdır. Yani, bu klasör altındaki tüm dosyalar aynı dil ile yazılmış olmalıdır. Nitekim, klasörümüze vb.net ile yazılmış aşağıdaki class dosyasını eklediğimizi düşünelim.

```
Imports Microsoft.VisualBasic
```

```
Public Class VbSinif
```

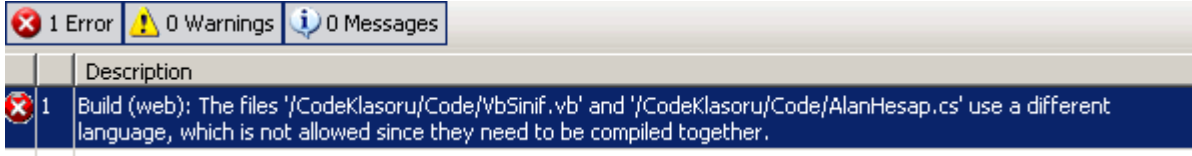
```
    Public Function Deneme(ByVal yarıcap As String) As String
```

```
        Return (yarıcap)
```

```
    End Function
```

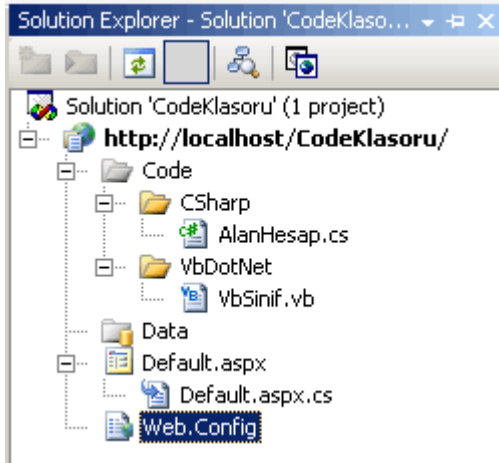
```
End Class
```

Bu durumda solution' ı derlediğimizde aşağıdaki derleme zamanı hata mesajını alırız.



Şekil 7. Hata Mesajı.

Peki çözüm nedir? Büyük çaplı projelerde, farklı .net dilleri kullanılarak geliştirilen sınıfların aynı solution içerisinde kullanılması için ne yapabiliriz? Bunun için, öncelikle Code klasörü içinde her bir dile yönelik olarak ayrı klasörler açmamız gerekir. Aşağıdaki şekilde olduğu gibi.



Şekil 8. Farklı diller için farklı klasörler.

Buradaki alt klasörleri isimlendirmek için belirli bir kural yoktur. Ancak buradaki isimlendirmelerin aynısını Web.Config dosyasındaki `<compilation>` node'unda kullanmamız gerekmektedir. Nitekim, otomatik olarak yapılan önceden derleme işlemlerinde, hangi alt klasörlerin kullanılacağını sitenin konfigürasyon ayarlarına yansıtılması gereklidir. Bunun için Web.Config dosyasındaki,

```
<compilation debug="true">
```

```
</compilation>
```

kısmını aşağıdaki ile değiştirmemiz yeterli olacaktır.

```
<compilation debug="true">
```

```
<codeSubDirectories>
```

```
<add directoryName ="CSharp"/>
```

```
<add directoryName ="VbDotNet"/>
```

```
</codeSubDirectories>
```

```
</compilation>
```

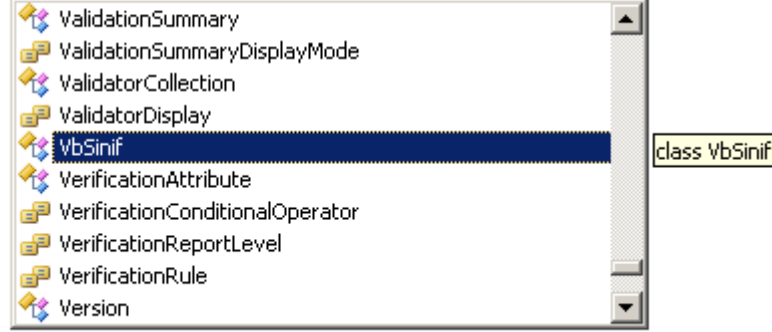
Şimdi bu işlemlerin ardından default.aspx sayfasına geçtiğimizde, vb.net ile yazdığımız sınıfa erişebildiğimizi ve kullanabildiğimizi görürüz.

```

void btnHesapla_Click(object sender, EventArgs e)
{
    AlanHesap ah = new AlanHesap();
    lblSonuc.Text = ah.DaireAlan(Convert.ToDouble(txtYaricap.Text)

    VbSinif s=new
}

```



Şekil 9. Farklı dil ile yazılmış sınıfa erişim.

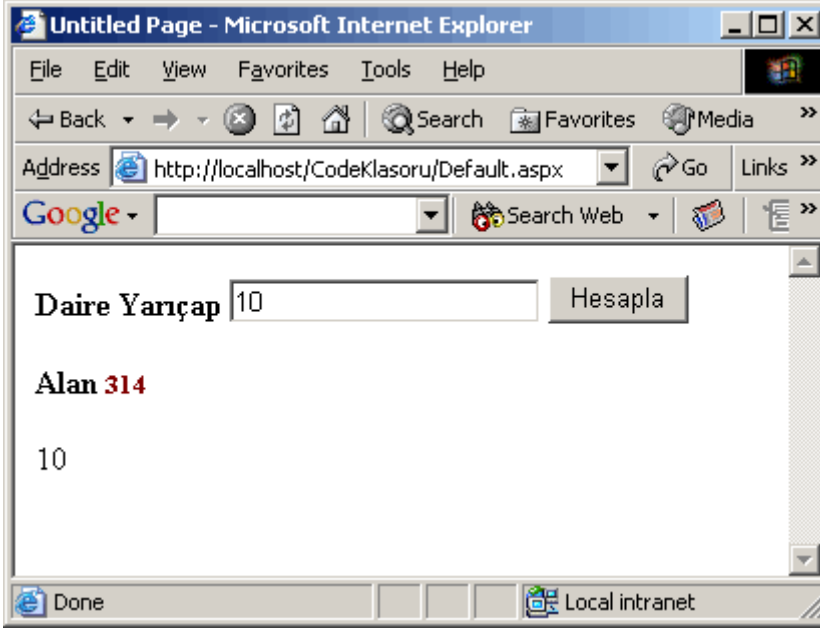
Şimdi kodumuzu aşağıdaki gibi geliştirelim ve sayfamızı çalıştıralım.

```

void btnHesapla_Click(object sender, EventArgs e)
{
    AlanHesap ah = new AlanHesap();
    lblSonuc.Text = ah.DaireAlan(Convert.ToDouble(txtYaricap.Text)).ToString();

    VbSinif s = new VbSinif();
    string deger = s.Deneme(txtYaricap.Text.ToString());
    lblDeger.Text = deger;
}

```



Şekil 10. Farklı dil ile yazılmış sınıflar bir arada çalışıyor.

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Asp.Net 2.0 ile Veri Kümelerinde Sayfalama İşlemleri

Değerli Okurlarım Merhabalar,

Bu makalemizde, Asp.Net 2.0 ile geliştirilen sayfalarda, veri kümeleri üzerinde sayfalama işlemlerinin nasıl yapıldığını incelemeye çalışacağız. Sayfalama işlemleri, özellikle internet(intranet) uygulamalarında yaygın şekilde kullanılan bir tekniktir. Burada, veri kümesine atı olan satırlar, DataGrid gibi bir kontrolde gösterilirken, sayfalara ayrılırlar. Böylece veri kümesine ait satırlar arasında toplu geçiş yapmamıza imkan sağlayan, navigasyon seçeneklerine sahip olmuş oluruz.

Asp.Net 2.0 içinde aynı imkanlar ve kabiliyetler söz konusudur. Ancak uygulanış şekli ve kullanılan bileşenler çok daha farklıdır. Herşeyden önce, Asp.Net 1.0/1.1 de izlenen yollara nazaran, daha kısa ve etkili bir teknik geliştirilmiştir. Asp.Net' in ilk sürümleri ile geliştirilen uygulamalarda, sayfalama işlemlerinin gerçekleştirilmesi için DataGrid kontrollerinde ekstradan olay prosedürü kodlamamız gerekmektedir. Bu bizim için fazla maliyettir. Nitekim zaman kaybettirici bir işlemdir. Ne demek istediğimi ve Asp.Net 2.0' da hangi noktaya geldiğimizi görmek için Visual Studio.Net 2003 ile geliştirilen aşağıdaki internet sayfasına bir göz atalım.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 1</title>
</head>
<body>
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false" Inherits="Sayfalama.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
```

```

<title>WebForm1</title>
<form id="Form1" method="post" runat="server">
<asp:DataGrid id="DataGrid1" runat="server" AllowPaging="True">
<PagerStyle Mode="NumericPages"></PagerStyle>
</asp:DataGrid>
</form>
</body>
</html>

```

Bu aspx sayfasında, DataGrid kontrolü üzerinde sayfalama işlemini gerçekleştirebilmek için AllowPaging özelliğine true değeri atanmıştır. Bununla birlikte sayfalamanın sayısal olarak gerçekleştirilmesi için PagerStyle takısı eklenmiş ve bu takının Mode özelliğine NumericPages değeri atanmıştır. Code-Behind kodlarımıza bakacak olursak, benzer uygulamalarda da muhtemelen aşağıdaki kod satırlarında kullanılan tarzda bir veri elde edilmiş yöntemi uygulandığını görürüz.

```

private void veriAl()
{
    SqlConnection con=new SqlConnection("data source=localhost;initial catalog=pubs;integrated security=SSPI");
    SqlCommand cmd=new SqlCommand("Select au_lname,au_fname,address From authors",con);
    SqlDataAdapter da=new SqlDataAdapter(cmd);
    DataTable dtYazarlar=new DataTable();
    da.Fill(dtYazarlar);
    DataGrid1.DataSource=dtYazarlar;
    DataGrid1.DataBind();
}

private void Page_Load(object sender, System.EventArgs e)
{
    if(!Page.IsPostBack)
    {
        veriAl();
    }
}

private void DataGrid1_PageIndexChanged(object source, System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
{
    DataGrid1.CurrentPageIndex=e.NewPageIndex;
    veriAl();
}

```

Sayfamızı bu haliyle çalıştırdığımızda, veri kümesi içerisinde sayfalama işlemlerinin gerçekleştirilebildiğini görürüz. Elde ettiğimiz küme, sayfalara ayrılarak, DataGrid kontrolünde gösterilecektir. Yaptığımız işlem aslında basittir ancak uzundur. Öncelikle bir SqlConnection kontrolü ile veri kaynağına bir bağlantı hattı çekilir. Ardından, ilgili veri kümesini elde edebileceğimiz, Select

sorgusunu taşıyan bir SqlCommand nesnesi oluşturulur. SqlCommand nesnesini kullanacak olan bir SqlDataAdapter ise, bağlantısız katmandaki DataTable nesnesini veri kümesi ile doldurmaktadır. Ardından elde edilen DataTable bileşeni DataGrid kontrolüne bağlanır ve DataGrid kontrolü için DataBind metod çalıştırılır. Tüm bu işlemlerin yanında sayfanın PostBack olması durumunda kaybolacak paging özelliğinin önüne geçmek için, Load olayında, IsPostBack kontrolü yapılır. Ayrıca, DataGrid kontrolünde, sayfalama işlemi sonrası oluşan sayfa numarası linklerine basıldığında, veri kümesinin ilgili parseline gidebilmek için, DataGrid1_PageIndexChanged olay metoduda kodlanmıştır. Bu yol her ne kadar bir süre Asp.Net ile uygulama geliştiren biris için anlaşılır ve kolay gözüksede, takdir edersinizki uzun ve aynı zamanda verimsiz bir yapıdır. Herşeyden önce, çok fazla kaynak tüketimi söz konusudur.

au_lname	au_fname	address
McBadden	Heather	301 Putnam
Ringer	Anne	67 Seventh Av.
Ringer	Albert	67 Seventh Av.
1	2	3

Şekil 1. Asp.Net 1.1 Sayfalama işlemi.

Burada takip ettiğimiz yolu düşüncük olursak, aslında gereksiz yere bir kaç adım işlem yaptığımızı ve bir kaç nesne kaynağını gereksiz yere harcadığımızı düşünebiliriz. Bu teknik dışında, SqlDataReader nesnesini kullanacağımız başka bir yol daha geliştirebilirdik. Ancak hangi yol seçilirse seçilsin, her ikisinde uygulama geliştiricinin bir kaç satırda olsa fazladan kod yazmasını ve bazı püf noktalara (DataGrid1_PageIndexChanged olay metodu gibi) dikkat etmesini gerektirecektir. Bu elbetteki uygulama geliştiricinin artan tecrübesi ile önemsiz hale gelebilir. Ancak işlerin dahada kısaltılarak yapılabileceğide gerçektir. İşte Microsoft mimarları, bu eksikliğin farkına varmış olacaklarki, Asp.Net 2.0' da, sayfalama işlemine farklı bir yaklaşım ve uygulama tekniği getirmişler. Herşeyden önce, DataSource kavramını kullanan Framework 2.0 için, sayfalama işlemlerini gerçekleştirmek, Asp.Net 1.0/1.1 sürümlerine göre hem daha kolay hemde daha profesyonel bir anlayışa sahip. Bu yeni teknik sayesinde, uygulama geliştiricinin verimliliğinin daha da artacağı kanısındayım. Şimdi dilerseniz, Asp.Net 2.0' daki duruma bir göz atalım. Bu kez aspx sayfamızın kodlarını aşağıdaki gibi oluşturacağız.

```
<%@ Page Language="C#" CompileWith="Sayfalama.aspx.cs" ClassName="Sayfalama_aspx" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
<head runat="server">
```

```
<title>Untitled Page</title>
```

```
</head>
```

```
<body>
```

```
<form id="form1" runat="server">
```

```
<div>
```

```

<asp:GridView ID="GridView1" Runat="server" AllowPaging="True" DataSourceID="sqlKaynak">
</asp:GridView>
<asp:SqlDataSource runat="Server" ProviderName="System.Data.SqlClient" ConnectionString="data source=.;initial
catalog=AdventureWorks;integrated security=true" SelectCommand="SELECT [EmployeeID],[Gender],[HireDate] FROM
[AdventureWorks].[HumanResources].[Employee]" ID="sqlKaynak"></asp:SqlDataSource>

</div>
</form>
</body>
</html>

```

Bu kodlara sahip Asp.Net 2.0 sayfamızı çalıştırdığımızda aşağıdaki gibi bir ekran görüntüsü elde ederiz.

EmployeeID	Gender	HireDate
61	F	20.01.1999 00:00:00
62	M	22.01.1999 00:00:00
63	M	23.01.1999 00:00:00
64	M	23.01.1999 00:00:00
65	M	24.01.1999 00:00:00
66	F	24.01.1999 00:00:00
67	M	25.01.1999 00:00:00
68	M	25.01.1999 00:00:00
69	F	26.01.1999 00:00:00
70	F	26.01.1999 00:00:00
1 2 3 4 5 6 7 8 9 10 ...		

Şekil 2. Asp.Net 2.0 için sayfalama işlemi.

Burada dikkat edecek olursanız tek satır uygulama kodu yazılmamıştır. Bunun yerine tüm işlemler, GridView ve SqlDataSource Asp.Net kontrolleri ile gerçekleştirilmiştir. Bu kontroller, Asp.Net 2.0 ile gelen sayısız yeni bileşenden sadece ikisidir. GridView kontrolümüzün en önemli özelliği, kendisine veri kaynağı olarak bir SqlDataSource nesnesini bildiren DataSourceID özelliğidir. Bu özellik ile, veri kümesini ala belirttiği veri kaynağından çekecek olan SqlDataSource kontrolünün ID değeri belirtilir.

```

<asp:GridView ID="GridView1" Runat="server" AllowPaging="True" DataSourceID="sqlKaynak">
</asp:GridView>

```

SqlDataSource bileşenimiz, uygulama geliştiricisinin verimliliğini arttıran yenilikler içerir. Asp.Net 1.1 ile yazdığımız bir önceki örneğin aksine, burada veri kaynağına bağlanma, veri kümesini çekme ve bunları ilgili kontrol ile ilişkilendirme işlemleri tek bir

bileşen içerisindeki özellikle yardımıyla gerçekleştirilebilmektedir. Bu noktada kontrol her ne kadar DataAdapter' ı andırsada çok daha farklı olduğunu ProviderName özelliğine bakarak bile anlayabiliriz.

```
<asp:SqlDataSource runat="Server" ProviderName="System.Data.SqlClient" ConnectionString="data source=.;initial catalog=AdventureWorks;integrated security=true" SelectCommand="SELECT [EmployeeID],[Gender],[HireDate] FROM [AdventureWorks].[HumanResources].[Employee]" ID="sqlKaynak"></asp:SqlDataSource>
```

SqlDataSource kontrolü ile, burada örnek olarak, Yukon (Sql Server 2005) üzerinde yer alan AdventureWorks veritabanındaki Employee tablosuna bağlanılmıştır. ProviderName özelliği, hangi veri sağlayıcının kullanılacağını belirtir. Biz burada doğal sql motorunu kullanmak istediğimizden, System.Data.SqlClient sınıfını kullandık. ConnectionString ile tahmin edeceğimiz gibi, veri kaynağına bir bağlantı hattı tahsis etmekteyiz. SelectCommand özelliği ilede, çalıştırmak istediğimiz Select sorgusunu tanımlıyoruz. SqlDataSource kontrolü burada GridView kontrolü ile beraber çalışmaktadır. Dolayısıyla, sayfalama işlemini gerçekleştirmek için, GridView kontrolünün AllowPaging özelliğine true değerini atamamız ve kullanılacak DataSource bileşenini belirtmemiz yeterlidir.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde kısaca Asp.Net 2.0 için sayfalama işlemlerinin nasıl gerçekleştirilebildiğini incelemeye çalıştık. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Asp.Net 2.0 için Site Map Kullanımı

Değerli Okurlarım Merhabalar,

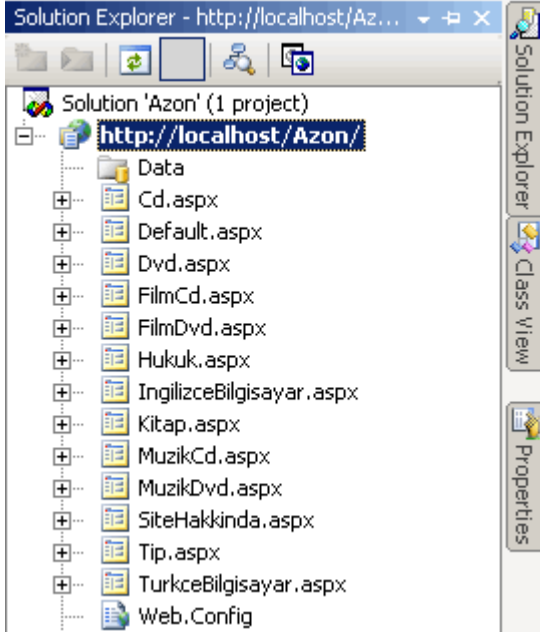
Bu makalemizde, web sitelerinde özellikle sayfalar arasındaki hareketlerde, kullanıcıların nerede olduklarını bilmelerine yardımcı olan site haritaları üzerinde duracağız. Bununla birlikte, site haritalarının Asp.Net 2.0' daki kullanım yollarından birisini sağlayan SiteMapPath sunucu kontrolünü kısaca incelemeye çalışacağız.

İnternet sitelerinin çoğu, pek çok web sayfasından oluşur. Bu sayfalar arasında her zaman için bir ilişki ve hiyerarşi söz konusudur. Dolayısıyla kullanıcılar, internet sitelerine ait sayfalarda gezinirken, onlara sitenin neresinde olduklarını göstermek ve bulundukları sayfaya nereden geldiklerini belirtmek, sitenin mantıksal ve anlamsal bütünlüğü açısından oldukça önemlidir. Bu tarz bir kolaylığı kullanıcılara sağlayabilmek amacıyla çeşitli yollar kullanabiliriz. Kullanıcı tanımlı kontroller geliştirebilir, programatik olarak geliştirmiş teknikler uygulayabiliriz. Ancak Asp.net 2.0, bu tip navigasyon izleme işlemleri için, Xml tabanlı, esnek ve daha pratik bir yol sunmaktadır. Bu teknikteki anahtar noktalar, Site Map dosyası ile, SiteMapPath sunucu kontrolüdür. Olayı daha iyi anlayabilmek amacıyla aşağıdaki hiyerarşik düzene sahip bir internet sitemizin olduğunu varsayalım.



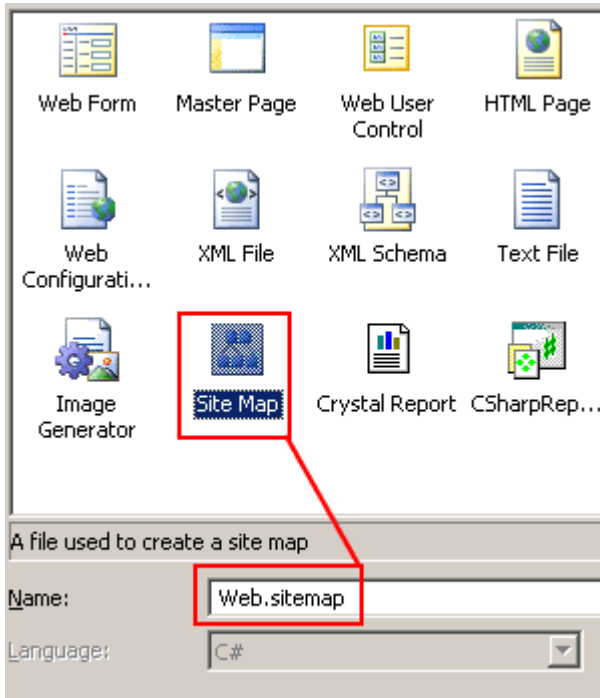
Şekil 1. Site Haritamız.

Burada görüldüğü gibi ana sayfamız dallanarak çeşitli alt sayfalara doğru ilerlenmektedir. Şimdi, Film Dvdsi sayfamızda olduğumuzu düşünelim. Burada kullanıcıya, nerede olduğu göstermek, son kullanıcı açısından oldukça değerli bir bilgidir. Heleki web sitemiz yüzlerce sayfa içeriyordu ve sayfaların çoğu 3 yada daha fazla alt sayfaya mantıki olarak bağlanıyorsa çok daha önemlidir. Ayrıca, kullanıcının bu sayfaya geldiği sayfalarada gitmesini kolayca sağlayacak linklerin olmasıda tercih nedenidir. İşte bu tip işlemleri gerçekleştirmek için, şekilsel olarak ifade ettiğimiz bu haritayı, Xml ortamına taşımamız gerekmektedir. Bu iş için kullanılan Site Map tipindeki dosyalar, Asp.Net 2.0 için geliştirilmiş özel nitelikli dosyalar olup tamamıyla, site haritalama işlemlerine hizmet etmektedir. Şimdi öncelikli olarak yukarıdaki hiyerarşiye sahip sitemizi Visual Studio.Net 2005 ortamında oluşturalım. Her sayfamızı aşağıdaki isimler ile Solution' ımıza ekleyelim.



Şekil 2. Azon internet sitemizin sayfaları.

Sıradaki işlemimiz web.sitemap dosyasını oluşturmak olacaktır. Bunun için Solution Explorer' da sağ tıklayıp Add New Item' i seçtikten sonra aşağıdaki şekilde görüldüğü gibi Site Map dosya tipini işaretlemeliyiz.



Şekil 3. Site Map dosyamızı oluşturuyoruz.

Gelelim dosyamızın içeriğine. Bu dosya Xml tabanlı bir dosya olup, siteMapNode takılarından oluşmaktadır. SiteMap takısı altında yer alan, siteMapNode takıları, site haritasındaki sayfaları çeşitli özellikleri ile birlikte belirtmektedir. Harita hiyerarşisine göre

derinlerdeki sayfalara inilmek istendiğinde iç içe geçen siteMapNode takıları kullanılmalıdır. siteMapNode takıları temel olarak, url, description, title olmak üzere 3 önemli özellik içerir. Title ile sayfanın başlığı, description ile sayfaya ait açıklama ve url özelliği ile sayfanın adresi belirtilmektedir. Şimdi, oluşturduğumuz web.sitemap dosyasına aşağıdaki Xml içeriğini yazalım.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<siteMap>
```

```
  <siteMapNode url="default.aspx" title="Azon Kitap Cd Dvd" description="En iyi Kitaplar, Cdler, Dvdler bu siteden alınır.">
```

```
    <siteMapNode url="kitap.aspx" title="Kitaplar" description="Bilgisayar, hukuk ve tıp kitapları">
```

```
      <siteMapNode url="IngilizceBilgisayar.aspx" title="Ingilizce Bilgisayar Kitapları" description="Ingilizce Bilgisayar Kitapları.
```

```
Onlardan iyisi yok."/>
```

```
      <siteMapNode url="TurkceBilgisayar.aspx" title="Türkçe Bilgisayar Kitapları" description="Türkçe Bilgisayar Kitapları
```

```
Zengin Arşivimiz ile Karşınızda."/>
```

```
      <siteMapNode url="Hukuk.aspx" title="Hukuk Kitapları" description="Hukuğu öğrenebileceğiniz en iyi kitaplar."/>
```

```
      <siteMapNode url="Tip.aspx" title="Tıp Kitapları" description="Tıp şakaya gelmez. En iyi tıp kitaplarını bizden alın."/>
```

```
    </siteMapNode>
```

```
  <siteMapNode url="cd.aspx" title="CD ler" description="Film ve Müzik Cd' leri.">
```

```
    <siteMapNode url="MuzikCd.aspx" title="Müzik Cd' leri." description="Harika gruplar, harika albümler."/>
```

```
    <siteMapNode url="FilmCd.aspx" title="Film Cd' leri." description="En güzel filmler."/>
```

```
  </siteMapNode>
```

```
  <siteMapNode url="dvd.aspx" title="DVD ler" description="Film ve Müzik DVD' leri.">
```

```
    <siteMapNode url="MuzikDvd.aspx" title="Müzik DVD' leri." description="Harika gruplar, harika albümler."/>
```

```
    <siteMapNode url="FilmDvd.aspx" title="Film DVD' leri." description="En güzel filmler çoklu dil destekleri ve mükemmel görüntü kalitelerine Dvdlerde."/>
```

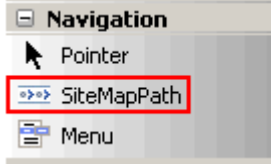
```
  </siteMapNode>
```

```
  <siteMapNode url="sitehakkinda.aspx" title="Site Hakkında" description="Kimiz ve size ne sağlıyoruz."/>
```

```
</siteMapNode>
```

```
</siteMap>
```

Artık elimizde site haritamızın xml içeriği mevcuttur. Şimdi bu Xml içeriğini kullanacak olan sunucu kontrolümüze bakalım. SiteMapPath sunucu kontrolü, bu xml içeriğine bakarak site haritasını değerlendirir ve son kullanıcıya akıllı navigasyon izleme yeteneğini sağlar.



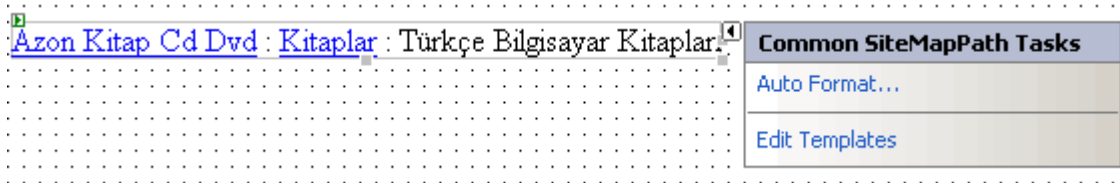
Şekil 4. SiteMapPath sunucu kontrolü.

Şimdi, toolbox' taki navigation sekmesinde yer alan SiteMapPath sunucu kontrollerini örnek olarak TürkceBilgisayar.aspx sayfasına bırakalım.

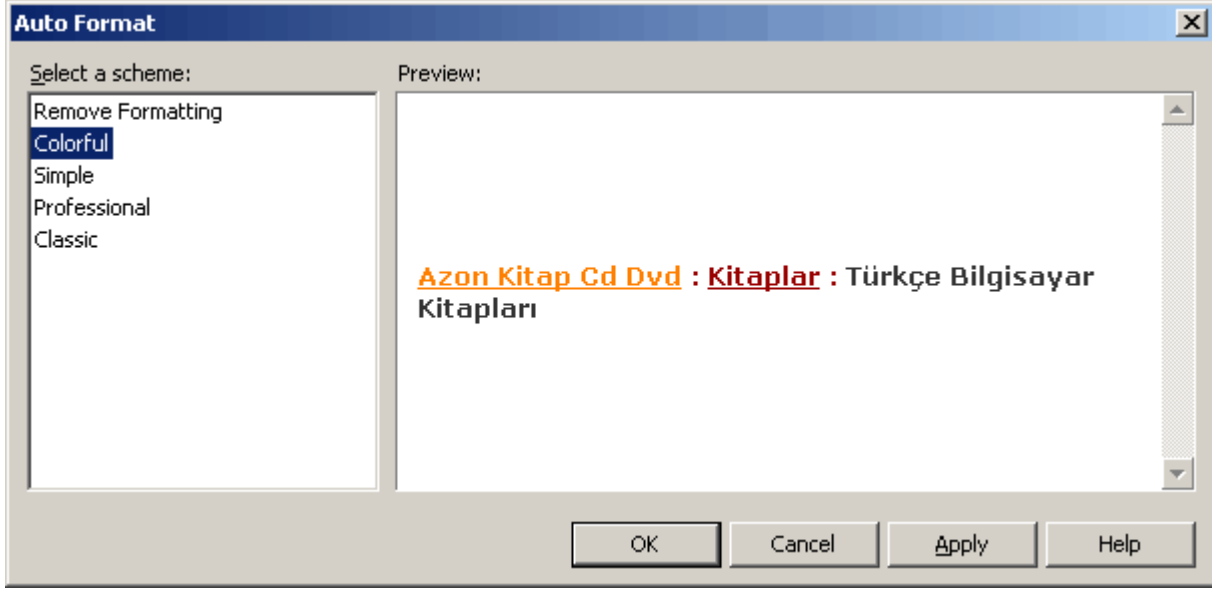


Şekil 5. Sunucu kontrolünün sayfaya eklenmesi sonrası.

Görüldüğü gibi, site haritamızdaki hiyerarşik düzene göre bu sayfaya hangi sayfalardan geldiği açıkça görülmektedir. Eğer bu noktada, TurkceBilgisayar.aspx sayfasını çalıştıracak olursak, "Kitaplar" ve "Azon Kitap Cd Dvd" linkleri ile, site haritasındaki üst sayfalara çıkılabileceğimizi de görürüz. Dilersek, SiteMapPath sunucu kontrolüne hazır görsel formatlardan birisini de uygulayarak makyajlayabiliriz. Bunun için kontrolün sağında çıkan ok işaretinden sonra yer alan Auto Format seçeneğine basmamız ve herhangi bir formatı (örneğin Colorful) seçmemiz yeterlidir.



Şekil 6. Auto Format seçeneği.



Şekil 7. Colorful seçeneği.

Bu makyajlama işlemi sonrası, SiteMapPath kontrolünün aspx sayfasındaki içeriği aşağıdaki gibi olacaktır.

```
<asp:SiteMapPath ID="SiteMapPath1" Runat="server" Font-Size="0.8em" Font-Names="Verdana"
```

```
PathSeparator=" : ">
```

```
  <PathSeparatorStyle Font-Bold="True" ForeColor="#990000"></PathSeparatorStyle>
```

```
  <CurrentNodeStyle ForeColor="#333333"></CurrentNodeStyle>
```

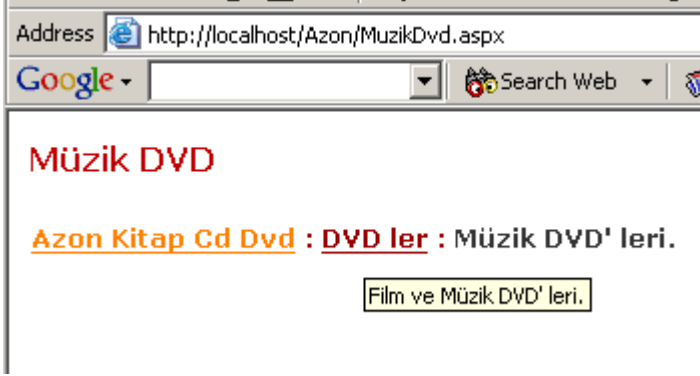
```
  <NodeStyle Font-Bold="True" ForeColor="#990000"></NodeStyle>
```

```
  <RootNodeStyle Font-Bold="True" ForeColor="#FF8000"></RootNodeStyle>
```

```
</asp:SiteMapPath>
```

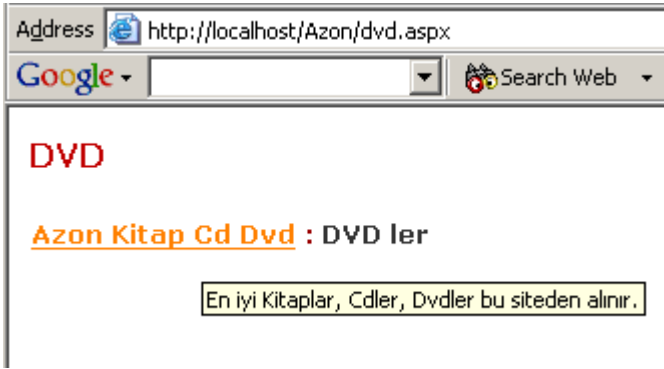
Burada önemli olan özelliklerden birisi, SiteMapPath sunucu kontrolüne ait, PathSeperator' dür. Bu özellik ile, sayfalar arasındaki ayraç işareti belirtilmektedir. Burada üst üste iki nokta kullanılmıştır. PathSeparatorStyle takısı ise, ayraç işaretinin font ve renk özelliklerini belirlemekte kullanılır. CurrentNodeStyle , kullanıcının o an bulunduğu sayfayı belirten node' un font ve renk özelliklerini belirlerken, NodeStyle üst nodelara ait ve RootNodeStyle ana sayfaya ait font ve renk özelliklerini belirler.

Şimdi, bu sayfaya uyguladığımız SiteMapPath sunucu kontrollünü kopyalayalım ve diğer sayfalarımıza yapıştıralım. Örnek olarak bu kezde, MuzikDvd.aspx sayfamızı çalıştıralım. Bu sayfayı çalıştırdığımızda ekran görüntüsü aşağıdaki gibi olacaktır.



Şekil 8. MuzikDvd sayfamız.

Görüldüğü gibi, Xml içeriğinde belirttiğimiz description değeri burada ilgili sayfa için bir ipucu kutucuğu olarak ekrana çıkmıştır. Diğer taraftan, altı çizili linklere tıklayarak üst sayfalara hareket edebiliriz. Örneğin, Dvd ler linkine tıkladığımızda, Dvd.aspx sayfasına gideriz.



Şekil 9. Dvd.aspx sayfasına geçtik.

SiteMapPath sunucu kontrolü için önemli olan özelliklerden birisi de, PathDirection' dır. Bu özellik, kontrol üzerindeki hiyerarşinin root' tan current'a yada tam tersi istikamette olup olmayacağını belirtir. Varsayılan olarak bir SiteMapPath kontrolünün yönü, root' tan current' a doğrudur. Eğer yönü ters çevirmek istersek, sunucu kontrolüne ait kodu aşağıdaki gibi değiştirmemiz gerekir.

```
<asp:SiteMapPath ID="SiteMapPath1" Runat="server" Font-Size="0.8em" Font-Families="Verdana"
PathSeparator=" : " PathDirection="CurrentToRoot">
```

Bu durumda, sunucu kontrolümüz aşağıdaki gibi görünecektir.



Şekil 10. PathDirection özelliği CurrentToRoot yapıldığında.

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde, TreeView kontrolü yardımıyla navigasyon işlemlerinin nasıl yapıldığını incelemeye çalışacağız. Şimdilik hoşçakalın.

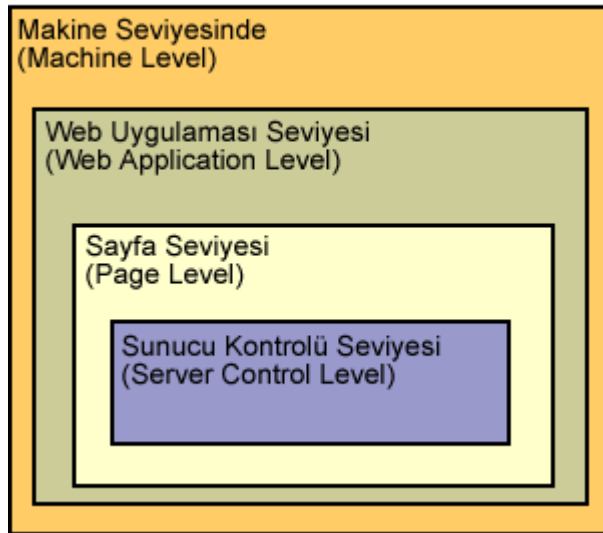
Asp.Net 2.0 ve Temalar (Themes)

Değerli Okurlarım Merhabalar,

Bir internet sitesini önemli kılan özellikler, sayfalarının font, renk, nesne yerleşimleri ile kontrollere ait görsel özellikleri açısından birbirleriyle olan uyumluluklarıdır. Bu anlamda, çoğu zaman sayfalara CSS stilleri uygulanır ve web uygulamasındaki tüm görsel öğelerin aynı font, renk vb özelliklere sahip olması sağlanır. Asp.Net 2.0, bu tarz stillerin çeşitli seviyelerde uygulanmasını sağlayacak yeni bir özellik sağlamaktadır. Themes (Temalar). İşte bu makalemizde, Asp.Net 2.0'da yer alan temaların, hangi seviyelere, nasıl ve ne şekilde uygulanabileceğini incelemeye çalışacağız.

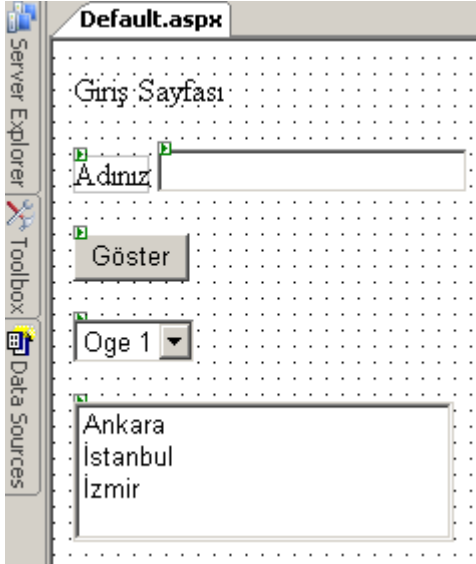
Bir tema, aslında işletim sistemlerinde uygulanan temalardan çok da farklı değildir. Pek çoğumuz kullandığı işletim sistemine temalar uygulamıştır. Uygulanan temalar, çoğunlukla sistemdeki font özelliklerini, pencerelere ait çeşitli renkleri, duvar kağıtlarını, sembolleri vb öğeleri görsel açıdan belirli bir standarta göre değiştirirler. Ancak internet projelerinin tasarlanması söz konusu olduğunda, temaların uygulanacağı seviyeler kavramı öncelikli olarak devreye girmektedir. Asp.Net 2.0 ile, makine seviyesinde, web uygulaması seviyesinde, sayfa seviyesinde ve sunucu kontrolü seviyesinde temalar uygulanabilmektedir.

Tema Seviyeleri (Themes Level)



Şekil 1 Tema Seviyeleri.

Biz bu makalemizde, temaların bu seviyelere nasıl uygulandığını incelemek amacıyla, basit bir tema oluşturacak ve bu temayı çeşitli seviyelere uygulayarak sonuçlarını incelemeye çalışacağız. Bu amaçla ilk olarak bir web uygulaması açalım. Bu web uygulamanızın default.aspx sayfasını aşağıdaki ekran görüntüsüne sahip olacak şekilde oluşturalım. Bu sayfada bir kaç sunucu kontrolünü kullanacağız.



Şekil 2. Sayfamızın Varsayılan Hali.

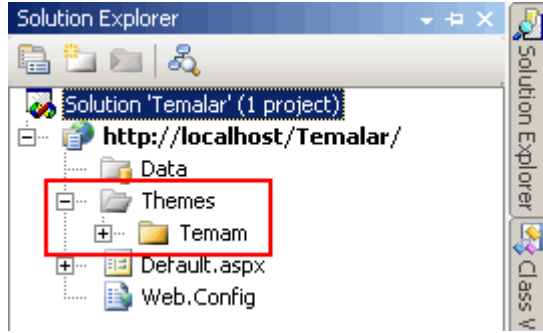
Eğer sayfamızı bu haliyle tarayıcıda açarsak aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 3. Sayfamızın ilk halinin tarayıcıdaki görüntüsü.

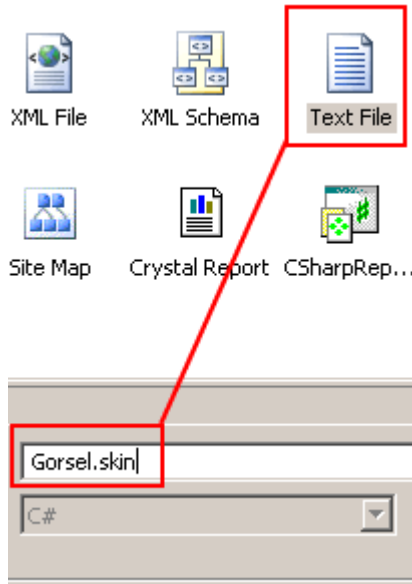
Şimdi uygulamamıza bir tema ekleyeceğiz. Bunun için öncelikle solution'ımıza Theme isimli bir folder eklememiz gerekiyor. Bu klasör, bir önceki makalemizde bahsetmiş olduğumuz Code klasörü gibi, solution için özel anlam ifade eden bir yapıya sahiptir. Nitekim bu klasör altına, uygulamaya ait font, renk gibi stillerin nasıl olacağını belirten css dosyaları ve sunucu kontrollerine ait görsel ayarların yapıldığı skin dosyaları ile çeşitli resimler eklenebilir. Theme klasörünü oluşturduktan sonra kullanacağımız tema için bir isim belirlemeli ve bu isimle yeni bir alt klasör açmalıyız. Artık skin ve css dosyalarımızı bu klasör altında oluşturabiliriz. Böylece,

uygulamamızda oluşturduğumuz alt klasör ismindeki temayı uygulayabiliriz. Klasör yapımız aşağıdaki şekildeki gibi olmalıdır. Bu hiyerarşik düzen aynı zamanda, bir solution için birden fazla tema oluşturulabileceğininide göstermektedir.



Şekil 4. Temamız için uygulamamız gereken klasör hiyerarşisi.

Şimdi sunucu kontrollerimizin nasıl görüneceğini belirleyeceğimiz özellikler taşıyacak bir skin dosyası oluşturacağız. Bu dosyamızı Temam klasörü altında geliştireceğiz. Skinler özellikle temalar için tasarlanmış tipteki dosyalardır. Ancak, Asp.Net 2.0' ın Beta sürümünde bu dosya tipi tanımlanmamıştır. Dolayısıyla, klasörümüz altında Add New Item seçeneği ile skin eklemek istiyorsak, Text File tipini seçmemiz ve dosya ismimizin uzantısı skin olacak şekilde belirtmemiz gerekmektedir.



Şekil 5. Skin eklenmesi.

Bir skin dosyası, aslında sunucu kontrollerinin görsel açıdan özelleştirilmiş halini içerir. Dolayısıyla, bir web uygulamasına bir temanın uygulanması sonucu, bu uygulamadaki sunucu kontrolleri için skin dosyaları içerisindeki sunucu kontrollerinin görsel özellikleri uygulanır. Bunu aşağıdaki kod satırlarından daha iyi anlayabiliriz. Aşağıdaki kod satırları, Gorsel.skin dosyamıza ait olup çeşitli sunucu kontrollerine görsel açıdan zenginlik katacak özelliklere sahiptir.

```
<asp:Label Runat="server" Font-Bold="False" Font-Names="Forte" Font-Size="Large" ForeColor="Red"></asp:Label>
<asp:TextBox Runat="server" Font-Names="Verdana" Font-Size="Small" ForeColor="#C000C0" BackColor="#FFC080"
BorderColor="#C00000" BorderStyle="Solid" BorderWidth="2px" ></asp:TextBox>
```

```
<asp:Button Runat="server" Font-Bold="True" ForeColor="White" BackColor="OrangeRed" BorderColor="#C04000"
BorderStyle="Solid" BorderWidth="3px" />
<asp:DropDownList Runat="server" Font-Names="Bookman Old Style" Font-Size="Small" BackColor="#FFE0C0" Font-
Italic="True"></asp:DropDownList>
<asp:ListBox Runat="server" Width="100px" Height="70px" Font-Bold="True" ForeColor="Gold"
BackColor="SteelBlue"></asp:ListBox>
```

Bu kodlara dikkatlice bakacak olursanız, Label, Button, TextBox, DropDownList, ListBox gibi kontrollerin çeşitli font, renk özellikleri ile tanımlanmış olduklarını göreceksiniz. İşte bu noktadan sonra, her hangibir tema seviyesinde, Temam temasını uygularsak, sunucu kontrolleri burada belirtilen görsel özelliklere sahip olacaktır. Bu dosyanın oluşturulması ile birlikte ilk temamızı hazırlamış oluyoruz. Şimdi bu temanın, bahsetmiş olduğumuz seviyeler için nasıl uygulanacağını inceleyelim.

Öncelikle sayfa seviyesinde bu temayı default.aspx için nasıl uygulayacağımıza bakalım. Bunun için, default.aspx sayfasının, page direktifine ait, Theme özelliğinden yararlanacağız. Tek yapmamız gereken, theme özelliğine, oluşturduğumuz temanın bulunduğu klasörün ismini atamak olacaktır. (Dolayısıyla temanın ismini.)

```
<%@ Page Language="C#" CompileWith="Default.aspx.cs" ClassName="Default_aspx" Theme="Temam" %>
```

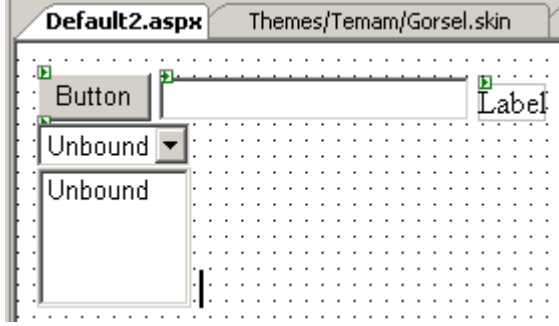
Şimdi default.aspx sayfamızın tarayıcıdaki görünümüne tekrar bakacak olursak, aşağıdaki ekran görüntüsünü elde ederiz.



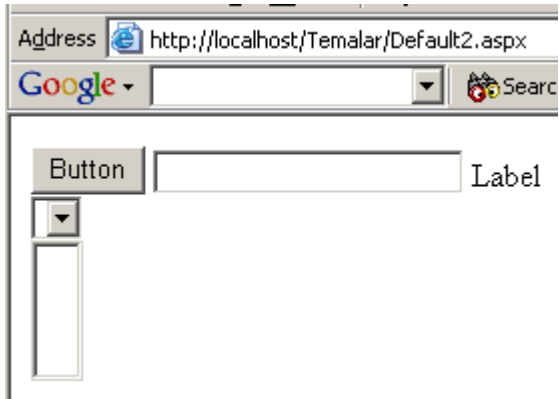
Şekil 6. Temamızın sayfaya uygulanmış hali.

Görüldüğü gibi, default.aspx çalıştığında, page direktifindeki theme özelliğinin sahip olduğu değerin belirttiği klasör, web uygulamasının Themes klasörü altında aranmış ve bulunduktan sonra, ilgili skin dosyasındaki kontrollere ait font ve renk özellikleri yüklenmiştir. Bu örneğimizde temamız, web sayfası seviyesindedir. Yani, uygulamamıza aşağıdaki ekran görüntüsüne sahip ikinci

bir form eklediğimizde, page direktifinin theme özelliğine temamızı atamadığımız sürece, sunucu kontrollerinin varsayılan görünümüne sahip olduğunu farkederiz.



Şekil 7. Theme özelliği ayarlanmamış sayfanın tasarım hali.

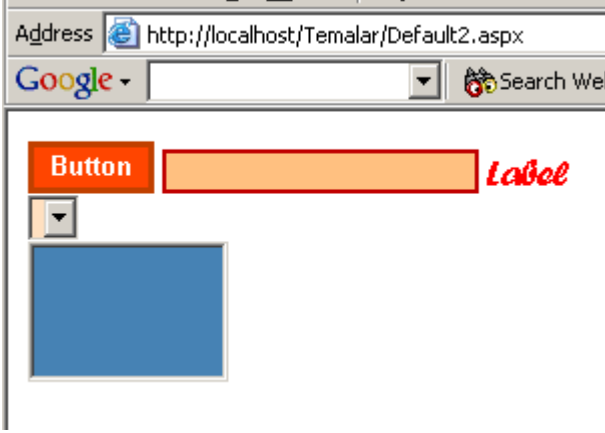


Şekil 8. Theme özelliği ayarlanmamış sayfanın çalışan hali.

Bu durumda karşımıza şöyle bir soru çıkar. Temalarımızı, web uygulamasının içerdiği tüm sayfalara uygulamak istersek ne olacak? İşte bu sorunun cevabı her zaman olduğu gibi, uygulamanın tamamına ait ayarlamaları yapabileceğimiz web.config dosyası ile ilgilidir. Sorunun çözümü için, web.config dosyasına pages takısını aşağıdaki syntax' ı ile eklememiz gerekir.

```
<system.web>
<pages theme="Temam"></pages>
.
.
.
</system.web>
```

Böylece, web uygulamasındaki tüm sayfaların, Temam isimli temayı uygulamasını sağlamış oluruz. Başka bir deyişle web uygulaması seviyesinde tema uygulatmış oluruz. Bu noktadan sonra default2.aspx sayfamızın çalışmasına bakarsak, sunucu kontrollerinin temayı uyguladığını görürüz.



Şekil 9. Uygulama seviyesinde tema kullanılmasının sonucu.

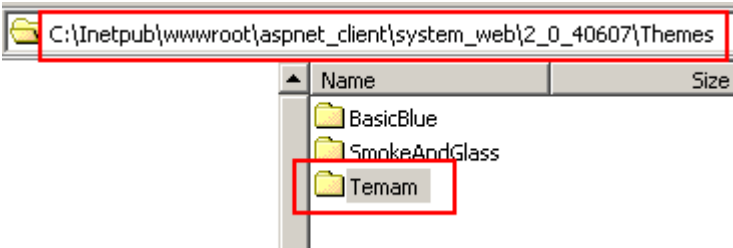
Diğer bir seviye olarak, makine seviyesinin olduğundan bahsetmiştik. Makine seviyesinde yapılan tema uyarlamaları, sistemde geliştirilen tüm web uygulamalarına, bu uygulamalardaki tüm web sayfalarına ve bu sayfalardaki tüm sunucu kontrollerine uygulanmaktadır. Makine seviyesinde tema uyarlamasını gerçekleştirmek için, windows 2000 işletim sistemleri için C:\WINNT\Microsoft.NET\Framework\v2.0.40607\CONFIG adresinde yer alan machine.config dosyasında yine pages takısındaki theme özelliğini aşağıdaki syntax' ta olduğu gibi ayarlamamız gerekmektedir.

```
<pages theme="Temam">
```

```
.  
.
.
```

```
</pages>
```

Ancak burada dikkat etmemiz gereken özel bir nokta vardır. Machine.config doysasına ekelenen temanın(temaların) sistemdeki tüm web uygulamalarına uyarlanabilmesi için C:\Inetpub\wwwroot\aspnet_client\system_web\2_0_40607\Themes adresi altında oluşturulması veya var olan temaların buraya kopyalanması gerekmektedir. Temam isimli klasörümüzü içeriği ile birlikte buraya kopyalarsak ve machine.config' de yukarıda bahsettiğimiz ayarlamayı yaparsak, sistemdeki tüm web uygulamalarının bu temayı uyguladığını görürüz.



Şekil 10. Temamızı global kullanıma açtığımız klasör.

Temalar ile ilgili önemli bir diğer noktada, üst seviyedeki temaların geçersiz kılınması işlemidir. Makine seviyesinde tema uyguladığımızı düşünürsek, herhangi bir web uygulamasında bu temayı uygulamak istemiyebiliriz. Bu durumda tek yapmamız gereken, web.config dosyasındaki page takısının theme özelliğine ya başka bir temayı atamak yada boş bırakmak olacaktır.

```
<pages theme=""></pages>
```

Böylece web uygulaması, makine seviyesinde belirtilen temayı geçersiz kılacaktır. Eğer sayfa seviyesinde, uygulama seviyesindeki yada makine seviyesindeki bir temayı geçersiz kılmak istersek, bu kez ilgili sayfanın page direktifindeki theme özelliğinde ya başka bir temayı belirtmemiz yada EnableTheming özelliğine false değerini vermemiz gerekecektir.

```
<%@ Page Language="C#" EnableTheming="false"%>
```

EnableTheming özelliği, sunucu kontrolleri içinde geçerlidir. Dolayısıyla, bir temayı uygulayan bir web sayfasındaki herhangi bir kontrol için tema özelliklerini geçersiz kılabiliriz. Bunun için tek yapmamız gereken sunucu kontrolünün EnableTheming özelliğine false değerini atamaktır. Örneğin, geliştirdiğimiz web uygulamasındaki Temam temasını göz önüne alalım. Eğer button kontrolünün EnableTheming özelliğini false yaparsak sayfamızı çalıştırdığımızda aşağıdaki ekran görüntüsünün oluştuğunu ve Button kontrolüne tema özelliklerinin uygulanmadığını görürüz.

```
<asp:Button ID="Button1" Runat="server" Text="Göster" EnableTheming="false"/>
```



Şekil 11. Bir kontrol için temanın geçersiz kılınması.

Böylece Asp.Net 2.0 ile gelen tema tekniğinin nasıl kullanıldığını kısaca incelemiş olduk. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

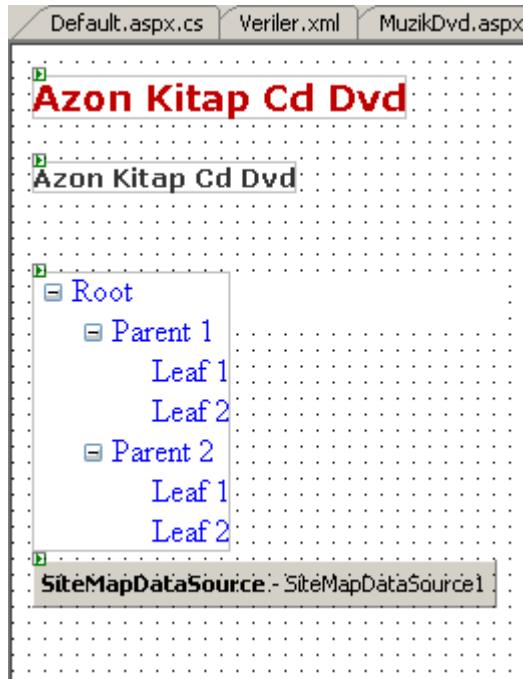
Asp.Net 2.0 ve TreeView Kontrolü

Değerli Okurlarım Merhabalar,

Bu makalemizde, Asp.Net 2.0 ile birlikte gelen yeni kontrollerden birisi olan TreeView kontrolü ile, özellikle Xml tabanlı veri kaynaklarına ait bilgilerin, internet ortamında hiyerarşik bir yapıda nasıl gösterilebileceğini incelemeye çalışacağız. Bir önceki makalemizden hatırlayacağınız gibi, site içinde son kullanıcıya akıllı navigasyon hizmeti sunabilmek için sitemap dosyalarından

faaydalanabileceğimizi ve bu dosyalara ait Xml içeriğinin, SiteMapPath kontrolü ile sayfalarımızda gösterilebileceğinden bahsetmiş ve basit bir örnek geliştirmiştik. Benzer işlevselliğe TreeView kontrolü yardımıyla daha kuvvetli bir biçimde sahip olabiliriz.

TreeView kontrolü, Xml tabanlı veri kaynaklarını kullanır. Bu anlamda, hiyerarşik verilerin özellikle ağaç yapısı şeklinde ifade edilmesinde kullanılmaktadır. Şimdi bir önceki makalemizde geliştirdiğimiz örneğe TreeView kontrolünü ekleyerek, sitemap işleminin nasıl yapılacağını incelemeye çalışalım. Öncelikle, formumuza bir TreeView kontrolü ve birde SiteMapDataSource kontrolü ekleyeceğiz. TreeView kontrolünün önemli bir özelliği mutlaka bir veri kaynağına bağlanması gerekliliğidir. Oysaki SiteMapPath kontrolü doğal olarak, site için oluşturulan sitemap dosyasına doğrudan bağlanabilmektedir. Fakat TreeView kontrolü Xml tabanlı verilere bağlanabilirliğinin gücünü yansıtabilmek için, DataSource kontrollerinden birisini kullanmak zorundadır. Bizim burada kullanacağımız SiteMapDataSource basit bir şekilde, sitenin sitemap dosyasına bağlanır ve Xml tabanlı içeriğin node seviyesinde, TreeView kontrolü ile ilişkilendirilmesini sağlar. İlk olarak bu kontrolleri geçen örneğimizdeki default.aspx sayfasına aşağıdaki tarzda yerleştirelim.

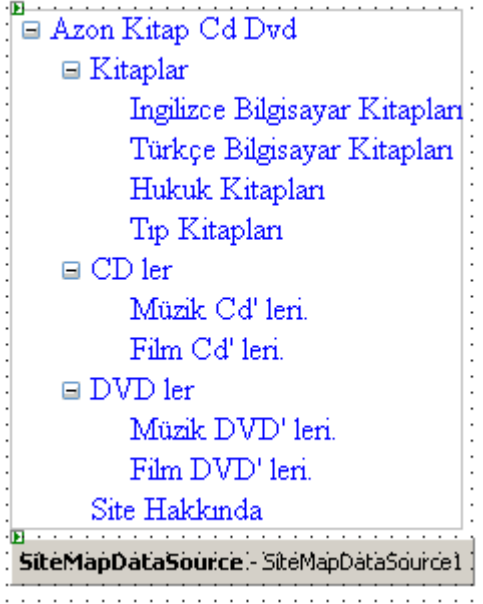


Şekil 1. Form Tasarımımız.

Formumuzu yukarıdaki gibi tasarladıktan sonra tek yapmamız gereken, TreeView kontrolüne veri kaynağını bildirmek olacaktır. Burada veri kaynağımız SiteMapDataSource kontrolü olduğu için, TreeView kontrolünün DataSourceID özelliğine, bu kontrolün ID değerini atamamız yeterlidir. Bu atamayıda yaptığımız takdirde aspx kodlarımız aşağıdaki gibi olacaktır.

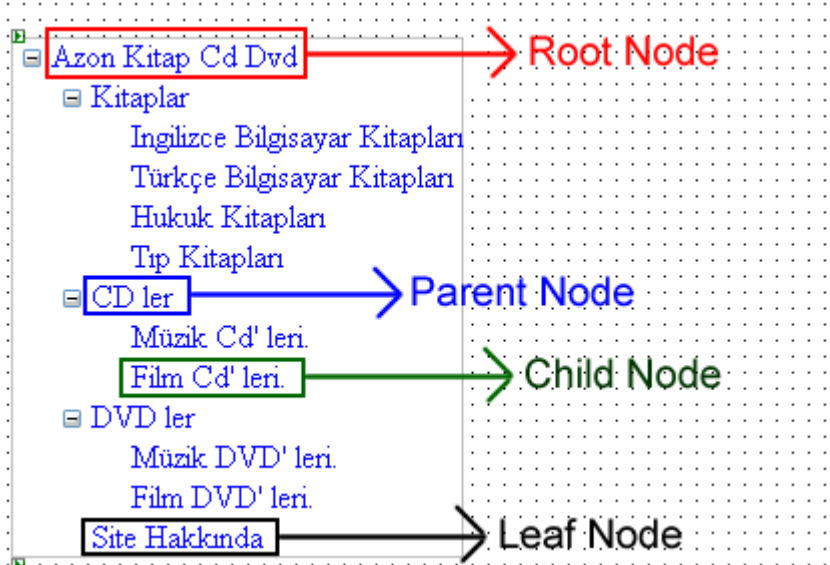
```
<asp:TreeView ID="TreeView1" Runat="server" DataSourceID="SiteMapDataSource1"></asp:TreeView>
<asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />
```

Ayrıca, tasarım ekranımızda aşağıdaki görünümü alacaktır.



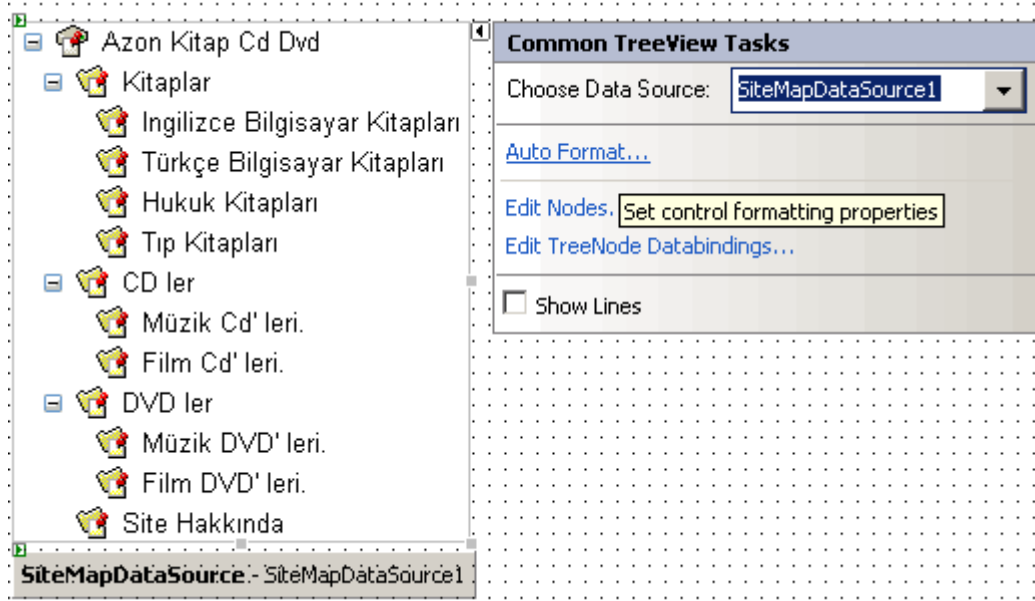
Şekil 2. TreeView, SiteMapDataSource kontrolüne bağlandıktan sonra.

Görüldüğü gibi, sitemap dosyasının Xml içeriği, SiteMapDataSource kontrolü yardımıyla, TreeView kontrolüne bağlanmıştır. Böylece hiyerarşik yapının ağaç görünümü elde edilmiştir. Bu noktada, ağaç görünümünün temel yapıtaşı olan node kavramından bahsetmekte yarar olacağını düşünüyorum. TreeView kontrolünün ifade ettiği ağaç yapısında, "Azon Kitap Cd Dvd" elemanı, root node olarak adlandırılır. Root Node, başka alt node' lar içerebilen ve hiyerarşide en üst sırada yer alan node olarak değerlendirilir. Bununla birlikte bir TreeView kontrolü birden fazla Root Node içerebilmektedir. Root Node altında yer alabilecek 3 tip node daha vardır. Bunlar Parent, Child ve Leaf Node' lardır. Bir Parent Node, Child Node' lar içerir ve root node altında yer alır. Diğer taraftan, herhangi bir Child Node içermeyen node' lar Leaf Node olarak adlandırılır. Node' ların mantığını okuyarak anlamak her ne kadar karışık görünsede, aşağıdaki şekil bu konuyu daha net bir şekilde açıklamaktadır.



Şekil 3. Node Kavramı.

Tabi burada dikkat edecek olursanız, örneğin "CD ler" node' u Parent Node olmakla birlikte, Root Node' un Child Node' u olarakta değerlendirilir. Benzer şekilde, "Film CD'leri" node'u "CD ler" node' unun Child Node' u olmakla birlikte, başka alt node' lar içermediği için aynı zamanda bir Leaf Node' dur. Node' lar ile ilgili bu kısa açıklamalardan sonra, uygulamamızı geliştirmeye devam edelim. TreeView kontrolü, kullanıcıya görsel açıdan çeşitli imkanlarda beraberinde getirmektedir. Örneğin MSDN' de kullanılan ağaç yapısının formatını TreeView kontrolünüze uygulayabilirsiniz. Ya da, aşağıdaki ekran görüntüsünü veren, News formatında kullanabilirsiniz. Bunun için tek yapmanız gereken, TreeView kontrolünün Auto Format özelliklerini, önceden tanımlanmış formatlardan birini seçerek tamamlamaktır.



Şekil 4. Önceden tanımlanmış News formatının TreeView kontrolüne uygulanmasının sonucu.

Dilerseniz sayfamızı çalıştıralım ve TreeView kontrolünün bize nasıl bir imkan sağladığına bakalım. Sayfamızı çalıştırdığımızda tarayıcı penceresinde TreeView kontrolünün, sitemap dosyasının içeriğini, ağaç yapısı şeklinde aşağıdaki şekilde olduğu gibi gösterdiğini görürüz.



Şekil 5. Sayfamız çalıştığında.

İşin en güzel yanı, node' lardan birisine tıkladığımızda, sitemap dosyasında o node için belirtilen link' e başka bir deyişle sayfaya gidebilmemizdir. Örneğin, Muzik Cd' leri sayfasına tıkladığımızda, MuzikCd.aspx sayfasına gideriz. Dolayısıyla TreeView kontrolünü, onu sitemap dosyasına bağlayan SiteMapDataSource kontrolü ile birlikte bir kullanıcı tanımlı kontrol olarak geliştirmemiz halinde, sitemizin tüm sayfalarında son derece etkin ve şık bir navigasyon sistemine sahip olabiliriz.

TreeView kontrolünün bu kullanımı dışında daha pek çok tarz uygulaması vardır. Öncedende söylediğimiz gibi, bir TreeView kontrolü mutlaka suretle bir veri kümesine bağlanır. Az önceki örneğimizde, bu veri kaynağı, sitemap dosyamızdı ve buraya bağlanmak için SiteMapDataSource kontrolünü kullanmıştık. Dilersek bir TreeView kontrolünü, bir Xml dosyasına da bağlayabiliriz. Örneğin aşağıdaki Xml dosyasını göz önüne alalım. (Bu dosyayı, Veriler.xml adıyla Solution' ımıza ekledik.)

```
<?xml version="1.0" encoding="utf-8" ?>
<Manav>
  <Meyve Kategori="Elma">
    <Miktar Deger="1 Kilo"></Miktar>
```

```

    <Miktar Deger="2 Kilo"></Miktar>
    <Miktar Deger="3 Kilo"></Miktar>
  </Meyve>
  <Meyve Kategori="Kiraz">
    <Miktar Deger="1 Kilo"></Miktar>
    <Miktar Deger="2 Kilo"></Miktar>
  </Meyve>
  <Meyve Kategori="Karpuz">
    <Miktar Deger="5 Kilo"></Miktar>
    <Miktar Deger="5 - 10 Kilo"></Miktar>
    <Miktar Deger="2 - 5 Kilo"></Miktar>
  </Meyve>
  <Meyve Kategori="Limon">
    <Miktar Deger="3 Tane"></Miktar>
    <Miktar Deger="10 Tane"></Miktar>
    <Miktar Deger="10-> Tane"></Miktar>
  </Meyve>
</Manav>

```

Bu Xml dosyasını,TreeView kontrolümüze bağlayabilmemiz için, bir XmlDataSource kontrolünü kullanmamız gerekiyor. XmlDataSource kontrolümüzün, bu Xml dosyasına işaret edebilmesi için tek yapmamız gereken, DataFile özelliğine ilgili Xml dosyasının adresini belirtmek olacaktır. Bunun için aspx kodlarımızı aşağıdaki gibi geliştirmemiz yeterli olacaktır.

```

<asp:XmlDataSource ID="XmlDataSource1" Runat="server" DataFile="Veriler.xml">
</asp:XmlDataSource>

```

Şimdi, bu XmlDataSource nesnesini kullanarak, veriler.xml dosyasına bağlanacak ve içeriğini gösterecek bir TreeView kontrolünü sayfamıza ekleyelim. Bu kez, SiteMapDataSource kontrolünde olduğu gibi, TreeView kontrolünün DataSourceID özelliğine, XmlDataSource kontrolünün ID değerini atamamız tek başına yeterli olmayacaktır. Nitekim, Xml dosyasındaki node' ları, TreeView kontrolüne bind etmemiz gerekmektedir. Bunun için, TreeView kontrolünün <DataBindings> takısı kullanılır. Bu takı altında, her bir node için gerekli olan TreeNodeBinding aspx nesneleri tanımlanmalıdır. Bu bilgileri göz önüne alırsak, TreeView kontrolümüzü ve XmlDataSource kontrolümüzü aşağıdaki aspx kodları ile oluşturmamız gerekmektedir.

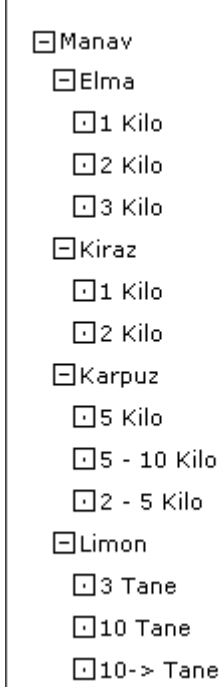
```

<asp:TreeView ID="trvManav" Runat="server" DataSourceID="XmlDataSource1">
  <DataBindings>
    <asp:TreeNodeBinding DataMember="Manav" Text="Manav" Value="Manav"></asp:TreeNodeBinding>
    <asp:TreeNodeBinding DataMember="Meyve" TextField="Kategori"></asp:TreeNodeBinding>
    <asp:TreeNodeBinding DataMember="Miktar" TextField="Deger"></asp:TreeNodeBinding>
  </DataBindings>
</asp:TreeView>

```

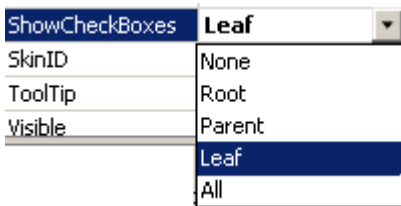
```
<asp:XmlDataSource ID="XmlDataSource1" Runat="server" DataFile="Veriler.xml">
</asp:XmlDataSource>
```

Burada bizim için önemli olan özellikler, TreeNodeBinding kontrolüne ait olan, DataMember ve TextField özellikleridir. Şimdi, TreeView kontrolümüze yine otomatik formatlardan birisini (örneğin MSDN) uygulayalım ve sayfamızı tarayıcıda açalım.



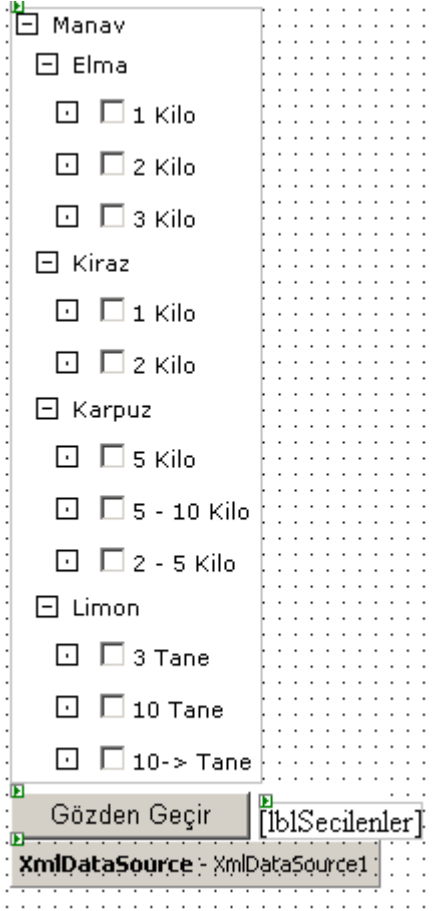
Şekil 6. TreeView kontrolünü Xml dosyasına, XmlDataSource kontrolü ile bağladığımızda.

Elbette, bu TreeView yapısını daha etkin bir şekilde kullanıma sunabiliriz. Örneğin, kullanıcı burada yer alan Leaf Node' lardan seçim yapabilir. Bunun için, TreeView kontrolünün, Leaf Node' larına CheckBox koymamızı sağlayacak bir özelliği vardır.



Şekil 7. ShowCheckBoxes özelliği.

ShowCheckBoxes özelliği ile, TreeView kontrolünün hangi node' larında CheckBox gösterileceğini belirtebiliriz. Biz bu örneğimizde leaf seçeneğini işaretleyelim. Böylece, kullanıcının hangi meyveden kaç kilo (adet) almak isteyeceğini işaretleyebileceği kutucuklarımız olacaktır. Kullanıcının, işaretlediği kutucukları elde edebilmek ve bir label kontrolünde gösterebilmek amacıyla, örneğimizi aşağıdaki şekilde olduğu gibi geliştirelim.



Şekil 8. Formun son hali.

Kullanıcı, sayfada "Gözden Geçir" başlıklı button kontrolüne tıkladığında, TreeView kontrolüne işaretlemiş olduğu miktarlar ve bu miktarların yer aldığı Parent Node' lar lblSecilenler isimli Label kontrolüne yazdırılacak. Bu işlemi gerçekleştirmek için, Button' umuzun Click olayına aşağıdaki kodları yazmamız yeterli olacaktır.

```
void btnGozdenGecir_Click(object sender, EventArgs e)
{
    string secilenler="";
    if (trvManav.CheckedNodes.Count > 0)
    {
        foreach (TreeNode tn in trvManav.CheckedNodes)
        {
            secilenler += tn.Parent.Text+" "+tn.Text+"|";
        }

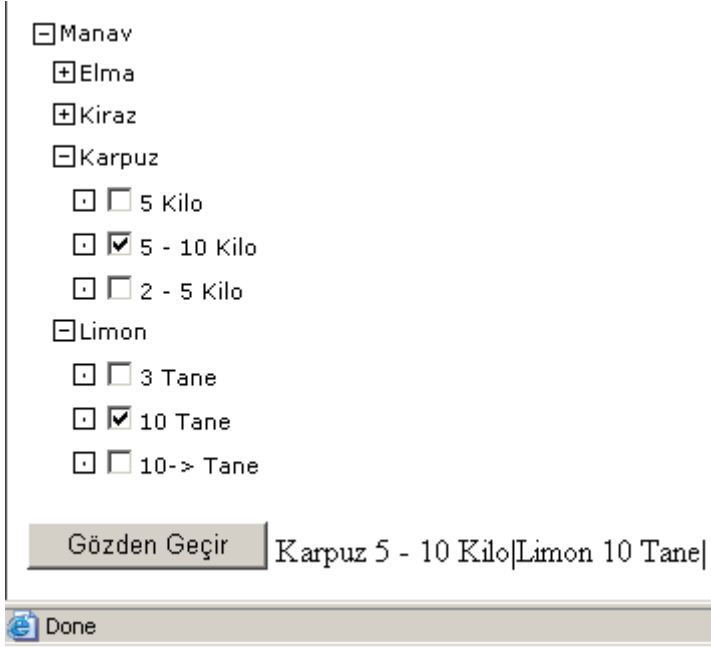
        lblSecilenler.Text = secilenler;
    }
}
```

```

else
{
    lblSecilenler.Text = " Manavdan bir şey almadınız...";
}
}

```

TreeView kontrolünün CheckedNodes özelliği TreeNodeCollections koleksiyonu tipinden değer döndürmektedir ve işaretlenmiş olan node' lar kümesini temsil etmektedir. Dolayısıyla if kontrolünde, Count özelliği ile, kullanıcının herhangi bir node' u işaretleyip işaretlenmediği, başka bir deyişle CheckBox kutucuklarını seçip seçmediğine bakılmaktadır. Eğer seçilmiş node' lar var ise, bu node' lar arasında gezinmek için, foreach döngüsü kullanılarak CheckNodes koleksiyonundaki her bir TreeNode elemanı ele alınır. TreeNode elemanları bu döngüde, Leaf Node' larımızdan işaretlenmiş olanlarına işaret etmektedir. Dolayısıyla, o anki TreeNode nesnesinin Parent özelliği, Parent Node' u işaret eder. Uygulamamızı çalıştırıp aşağıdaki gibi bir kaç seçim yaptığımız takdirde, Label kontrolümüzün seçili olan Leaf Node' lar ve bu node'ların Parent Node' ları ile doldurulduğunu görürüz.



Şekil 9. Sayfamız çalıştığında.

Böylece geldik bir makalemizin daha sonuna. İlerleyen makalelerimizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Asp.Net 2.0 GridView Kontrolünde Update,Delete İşlemleri

Değerli Okurlarım Merhabalar,

Bu makalemizde, Asp.Net 2.0 ile gelen yeni kontrollerden birisi olan GridView kontrolü üzerinde, veri güncelleme ve veri silme gibi işlemlerin nasıl yapılacağını incelemeye çalışacağız. Asp.Net 2.0 ve dolayısıyla Framework 2.0, özellikle veri bağlı kontrollerde, yazılım geliştiricilerin sıklıkla yaptıkları rutin işlemlerin dahada kolaylaştırılmasına izin veren mimari yaklaşımları benimsemektedir. Bu makalemize konu olan GridView kontrolü internet sayfalarında verilerin eski DataGrid kontrolünde olduğu gibi ızgara formatında görünmesini sağlar. Bununla birlikte GridView kontrolü üzerinde sayfalama ve sıralama gibi işlemleri yapmak eskiye nazaran çok

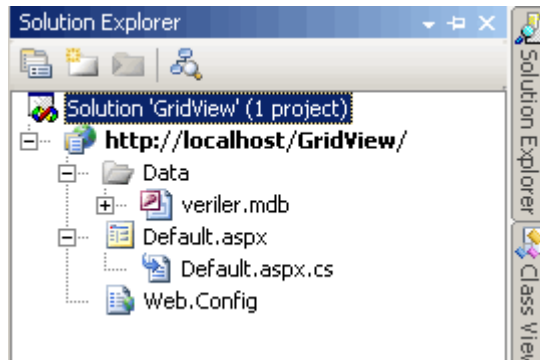
daha kolaydır. İşte GridView kontrolünün yazılım geliştirici dostu olan yapısının bir diğer kolaylığıda, satırlar üzerinde güncelleme ve silme işlemlerine getirdiği yeniliklerde gizlidir.

Bu makalemizde geliştireceğimiz örneğimizde, bir Access veri kaynağına bağlanacağımız bir web sayfamız olacak. Bu sayfada kullanacağımız GridView kontrolünü veri güncelleme ve silme işlemleri için kullanacağız. Bu örneği geliştirebilmemiz için, GridView kontrolünü Access veritabanımıza bağlayacak bir DataSource kontrolüne ihtiyacımız var. Access veritabanlarına bağlanmak ve ilgili veritabanı üzerindeki tablolarda işlem yapabilmek için, .net Framework 2.0 ile gelen yeni nesnelerden birisi olan AccessDataSource kontrolünü kullanacağız. Veritabanımız Access ile geliştirilmiş olup MailListesi isimli aşağıdaki Field yapısına sahip bir tabloyu barındırmaktadır.

MailListesi : Table		
	Field Name	Data Type
?	ID	AutoNumber
	Ad	Text
▶	Soyad	Text
	Mail	Text

Şekil 1. Access tablomuzun field yapısı.

Oluşturduğumuz bu tabloyu, web sitemiz için standart olarak açılan Data klasörü altına kopyalayım.



Şekil 2. Veritabanımızı Data klasörü altına koyalım.

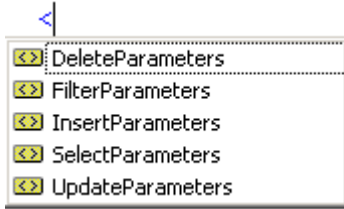
Sıra geldi default.aspx sayfamızı tasarlamaya. Öncelikli olarak, AccessDataSource kontrolümüzü aşağıdaki aspx kodları ile sayfaya ekleyelim.

```
<asp:AccessDataSource ID="AccessDataSource1" Runat="server" DataFile="~/Data/veriler.mdb"
SelectCommand="SELECT * FROM [MailListesi]" UpdateCommand="UPDATE [MailListesi] SET
Ad=@AD,Soyad=@SOYAD,Mail=@MAIL WHERE ID=@ID">
  <UpdateParameters>
    <asp:Parameter Name="AD"></asp:Parameter>
    <asp:Parameter Name="SOYAD"></asp:Parameter>
    <asp:Parameter Name="MAIL"></asp:Parameter>
    <asp:Parameter Name="ID"></asp:Parameter>
  </UpdateParameters>
```

```
</asp:AccessDataSource>
```

Burada dikkat ederseniz, SelectCommand özelliği dışında, dikkati çeken bir diğer özellik UpdateCommand' dir. UpdateCommand özelliği, satırlardaki değişikliklerin veri kaynağına yansıtılmasında kullanılacak Sql sorgusunu içermektedir. Bu sql sorgusu dikkat edecek olursanız, çeşitli parametrelere sahiptir. Dikkate değer diğer bir noktada, parametre bildirimlerinin, Framework 1.0/1.1' de olan, OleDbCommand kontrollerindeki gibi, ? (soru işareti) olmayışdır. Bu noktada, Sql sunucuları üzerinde çalıştırılan parametrelili sorgulardaki yapının (@Parametre_adı yapısı) aynısının Access kaynakları içinde kullanılabildiğini ve böylece ortak bir standardın oluşturulmuş olduğunda belirtelim.

UpdateCommand özelliğindeki Sql sorgusunda yer alan parametreler, AccessDataSource kontrolünün UpdateParameters koleksiyonunda birer Parameter nesnesi olarak tanımlanırlar. Basitçe burada parametrelerimizin isimlerini Name özellikleri ile belirledik. DataSource bileşenleri için, SelectCommand ve UpdateCommand komutları dışında kullanılabilecek diğer komutlarda, DeleteCommand ve InsertCommand' dir. Bunların herbirinin parametre koleksiyonları farklıdır. Yani DeleteCommand için parametre koleksiyonu DeleteParameters iken, InsertCommand için InsertParameters' dir.



Şekil 3. DataSource' lar için geçerli diğer parametre koleksiyonları.

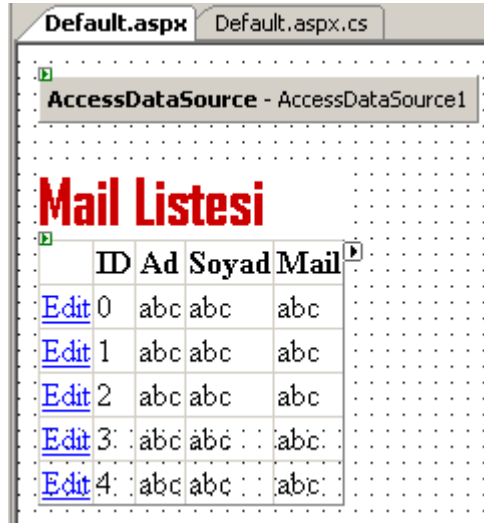
AccessDataSource bileşenimiz artık veri çekmek ve güncellemek için hazırdır. Sırada, GridView kontrolümüzü, bu veri kaynağına göre düzenlemek var. Bunun için, GridView kontrolümüzü aşağıdaki aspx kodları ile oluşturmamız yeterli olacaktır.

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="AccessDataSource1" DataKeyNames="ID"
AutoGenerateColumns="False">
  <Columns>
    <asp:CommandField ShowEditButton="true"></asp:CommandField>
    <asp:BoundField ReadOnly="True" HeaderText="ID" InsertVisible="False" DataField="ID"
SortExpression="ID"></asp:BoundField>
    <asp:BoundField HeaderText="Ad" DataField="Ad" SortExpression="Ad"></asp:BoundField>
    <asp:BoundField HeaderText="Soyad" DataField="Soyad" SortExpression="Soyad"></asp:BoundField>
    <asp:BoundField HeaderText="Mail" DataField="Mail" SortExpression="Mail"></asp:BoundField>
  </Columns>
</asp:GridView>
```

Şu aşamada bizim için en önemli özellik, DataKeyNames' dir. Bu özellik, GridView kontrolünde üzerindeki satırlarda yapılan veri değişiklikleri sonucu, güncelleme işlemlerinin hangi primary key alanı üzerinden yapılacağını belirtmektedir. Nitekim, Sql sorgumuza dikkat edecek olursanız, güncelleme işleminin hangi alan için yapılacağını belirleyebilmek amacıyla, primary key olan ID alanı kullanılmıştır.

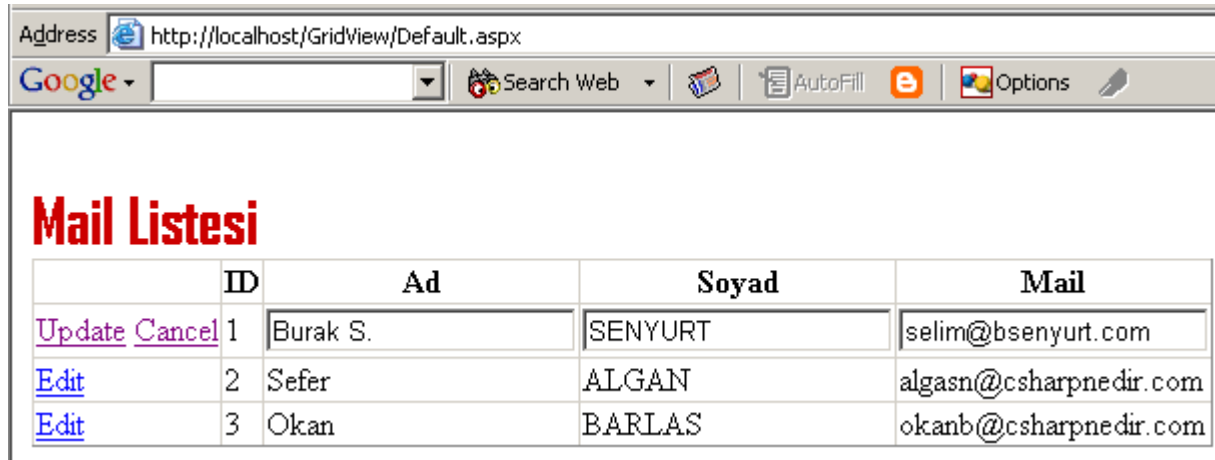

```
UPDATE [MailListesi] SET Ad=@AD,Soyad=@SOYAD,Mail=@MAIL WHERE ID=@ID
```

Bu sebeple, GridView kontrolü, Update butonuna basıldığında, DataKeyNames özelliğindeki field alanının değerini, seçili olan satır üzerinden AccessDataSource kontrolündeki UpdateCommand sorgusuna gönderecektir. Diğer önemli üye ise, CommandField kontrolüdür. Bu kontrol, satır üzerinde hangi işlemin yapılacağı ile ilgili link (button) kontrollerinin gösterilmesi için kullanılmaktadır. Burada ShowEditButton özelliğine true değerini vererek, ilgili satır için Edit, Update ve Cancel linklerinin gösterilmesini sağlamış olduk. Artık web formumuza son halini verebiliriz.



Şekil 4. default.aspx sayfamızın görünümü.

Şimdi sayfamızı tarayıcı penceresinde açar ve herhangi bir satırda edit linkine tıklayacak olursak, aşağıdaki görünümü elde ederiz.



Şekil 5. Edit linkine tıklandığında.

Görüldüğü gibi, ilgili satıra ait alanlarda TextBox kutucukları veri girişi için hazır beklemektedir. Bu noktada Update linkine tıklarsak, seçili satırın ID değeri ile birlikte diğer alanların güncel içerikleri, AccessDataSource kontrolünün UpdateCommand özelliğinde kullanılan Sql sorgusuna parametre olarak gönderilirler. Ardından güncelleme işlemi yapılır ve GridView kontrolü güncel içeriği ile

görüntülenir. Örneğin, "Burak S." değerini "Burak Selim" yapıp Update linkine bastığımızda aşağıdaki ekran görüntüsünü elde ederiz.



Mail Listesi

	ID	Ad	Soyad	Mail
Edit	1	Burak Selim	SENYURT	selim@bsenyurt.com
Edit	2	Sefer	ALGAN	algasn@csharpnedir.com
Edit	3	Okan	BARLAS	okanb@csharpnedir.com

Şekil 5. Güncelleme işlemi sonrası.

Tahmin edin, Delete işlemini nasıl gerçekleştireceğiz. Uygulayacağımız mantık aslında, Update işlemi için uygulanan ile aynı olacaktır. İlk olarak AccessDataSource kontrolümüzde, DeleteCommand özelliğini tanımlamalı ve geçerli bir Delete sql sorgusu girmeliyiz. Ardından bu Sql sorgusunda kullanacağımız ID alanı için bir parametre ekelemeliyiz. Yani AccessDataSource kontrolümüzü aşağıdaki hale getirmeliyiz.

```
<asp:AccessDataSource ID="AccessDataSource1" Runat="server" DataFile="~/Data/veriler.mdb"
SelectCommand="SELECT * FROM [MailListesi]" UpdateCommand="UPDATE [MailListesi] SET
Ad=@AD,Soyad=@SOYAD,Mail=@MAIL WHERE ID=@ID" DeleteCommand="DELETE FROM [MailListesi] WHERE ID=@ID">
  <UpdateParameters>
    <asp:Parameter Name="AD"></asp:Parameter>
    <asp:Parameter Name="SOYAD"></asp:Parameter>
    <asp:Parameter Name="MAIL"></asp:Parameter>
    <asp:Parameter Name="ID"></asp:Parameter>
  </UpdateParameters>
  <DeleteParameters>
    <asp:Parameter Name="ID"></asp:Parameter>
  </DeleteParameters>
```

Tabi GridView kontrolümüzde de Delete işlemi için gerekli linki (button' u) göstermeliyiz. Bu amaçla, GridView kontrolünün CommandField bileşeninin ShowDeleteButton özelliğinin değerini true yapmalıyız.

```
<asp:CommandField ShowDeleteButton="true" ShowEditButton="true"></asp:CommandField>
```

Bu değişikliklerden sonra sayfamızı çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz.

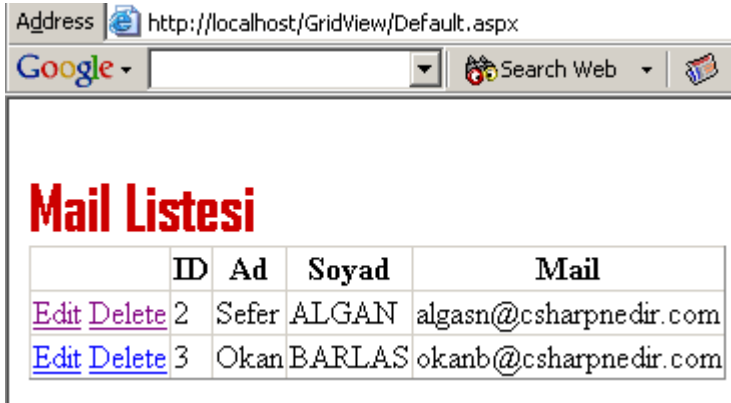


Mail Listesi

	ID	Ad	Soyad	Mail
Edit Delete	1	Burak Selim	SENYURT	selim@bsenyurt.com
Edit Delete	2	Sefer	ALGAN	algasn@csharpnedir.com
Edit Delete	3	Okan	BARLAS	okanb@csharpnedir.com

Şekil 7. Delete linkinin eklenmesi.

Eğer Delete linki ile bir satırı silecek olursak, ilgili satırın ID değeri, AccessDataSource kontrolündeki, DeleteCommand özelliğine parametre olarak gidecek ve Sql sorgusu çalıştırılarak ilgili satır silinecektir. Örnek olarak, 1 ID değerine sahip satırı sildiğimizde, aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 8. Satır silindikten sonra.

Bununla birlikte unutulmaması gereken önemli bir nokta vardır. Buradaki tüm işlemler veri tablosuna doğrudan yansıtılır. Dolayısıyla, satırı sildiğimizde(güncellediğimizde), AccessDataSource bunu kesinlikle, asıl tabloyada yansıtacaktır. Nitekim son işlemten sonra, tablomuza bakacak olursak ID değeri 1 olan satırın artık olmadığını görebiliriz.

MailListesi : Table				
	ID	Ad	Soyad	Mail
	2	Sefer	ALGAN	algasn@csharpnedir.com
	3	Okan	BARLAS	okanb@csharpnedir.com
*	(AutoNumber)			

Şekil 9. Satır kesin olarak silinir.

Geliştirdiğimiz örneği biraz makyajlamaya ne dersiniz? Örneğin, Update, Edit, Cancel ve Delete kelimeleri yerine Türkçe bir şeyler çaksa ve bunlar Link yerine button olsa fena olmazdı değil mi? Bunun için tek yapmamız gereken aşağıdaki aspx kodları ile,

GridView kontrolümüzü güncellemektir. Dikkat edecek olursanız, her bir kontrolün sonu Text kelimesi ile biten bir özelliği vardır. Örneğin Delete butonu için DeleteText. Dolayısıyla bunları değiştirerek ekranda görünmesini istediğimiz metni belirtebiliriz. Ayrıca, linklerimizi button haline getirmek için, ButtonType özelliğine Button değerini atadık.

```
<asp:CommandField ButtonType="Button" CancelText="İptal" EditText="Düzenle" DeleteText="Sil"
UpdateText="Güncelle" ShowDeleteButton="true" ShowEditButton="true"></asp:CommandField>
```

Mail Listesi

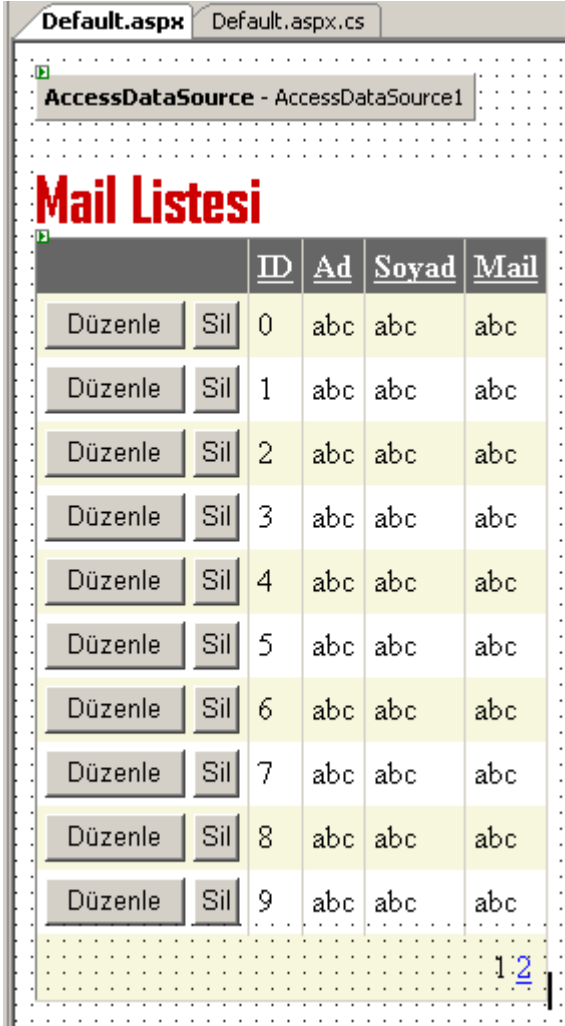
		ID	Ad	Soyad	Mail
Düzenle	Sil	2	Sefer	ALGAN	algasn@csharpnedir.com
Düzenle	Sil	3	Okan	BARLAS	okanb@csharpnedir.com

Mail Listesi

		ID	
Güncelle	İptal	2	Sefer
Düzenle	Sil	3	Okan

Şekil 10. Türkçe buttonlar.

Sayfamızın son olarak makyajlanmış hali ve ilgili aspx kodları aşağıdaki gibidir.



Şekil 11. Sayfamızın biraz daha makyajlı hali.

```
<%@ Page Language="C#" CompileWith="Default.aspx.cs" ClassName="Default_aspx" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
<head runat="server">
```

```
<title>Untitled Page</title>
```

```
</head>
```

```
<body>
```

```
<form id="form1" runat="server">
```

```
<div>
```

```
<asp:AccessDataSource ID="AccessDataSource1" Runat="server" DataFile="~/Data/veriler.mdb"
```

```
SelectCommand="SELECT * FROM [MailListesi]" UpdateCommand="UPDATE [MailListesi] SET
```

```

Ad=@AD,Soyad=@SOYAD,Mail=@MAIL WHERE ID=@ID" DeleteCommand="DELETE FROM [MailListesi] WHERE ID=@ID">
    <UpdateParameters>
        <asp:Parameter Name="AD"></asp:Parameter>
        <asp:Parameter Name="SOYAD"></asp:Parameter>
        <asp:Parameter Name="MAIL"></asp:Parameter>
        <asp:Parameter Name="ID"></asp:Parameter>
    </UpdateParameters>
    <DeleteParameters>
        <asp:Parameter Name="ID"></asp:Parameter>
    </DeleteParameters>
</asp:AccessDataSource><br/>
<b><span style="font-size: 24pt; color: #cc0000; font-family: Agency FB">Mail Listesi</span></b>
<asp:GridView ID="GridView1" Runat="server" DataSourceID="AccessDataSource1" DataKeyNames="ID"
AutoGenerateColumns="False" BorderWidth="1px" BackColor="White" GridLines="Vertical" CellPadding="4" BorderStyle="None"
BorderColor="#DEDFDE" ForeColor="Black" AllowSorting="True" AllowPaging="True">
    <FooterStyle BackColor="#CCCC99"></FooterStyle>
    <PagerStyle ForeColor="Black" HorizontalAlign="Right" BackColor="#F7F7DE"></PagerStyle>
    <HeaderStyle ForeColor="White" Font-Bold="True" BackColor="#6B696B"></HeaderStyle>
    <AlternatingRowStyle BackColor="White"></AlternatingRowStyle>
    <Columns>
        <asp:CommandField ButtonType="Button" CancelText="İptal" EditText="D&#252;zenle" DeleteText="Sil"
UpdateText="G&#252;ncelle" ShowDeleteButton="True" ShowEditButton="True"></asp:CommandField>
        <asp:BoundField ReadOnly="True" HeaderText="ID" InsertVisible="False" DataField="ID"
SortExpression="ID"></asp:BoundField>
        <asp:BoundField HeaderText="Ad" DataField="Ad" SortExpression="Ad"></asp:BoundField>
        <asp:BoundField HeaderText="Soyad" DataField="Soyad" SortExpression="Soyad"></asp:BoundField>
        <asp:BoundField HeaderText="Mail" DataField="Mail" SortExpression="Mail"></asp:BoundField>
    </Columns>
    <SelectedRowStyle ForeColor="White" Font-Bold="True" BackColor="#CE5D5A"></SelectedRowStyle>
    <RowStyle BackColor="#F7F7DE"></RowStyle>
</asp:GridView>
</div>
</form>
</body>
</html>

```

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Asp.Net 2.0 DetailsView Kontrolü ile Insert,Update,Delete

Değerli Okurlarım Merhabalar,

Bu makalemizde, Whidbey ile gelen yeni kontrollerden birisi olan DetailsView bileşeni ile, satır ekleme, silme ve güncelleme gibi işlemlerin nasıl yapıldığını incelemeye çalışacağız. Bununla birlikte öncesinde, DetailsView kontrolünü genel hatları ile kısaca araştıracağız. DetailsView kontrolü, Asp.Net 2.0' a eklenen yeni data-bound kontrollerinden birisidir. Bu kontrolün en büyük özelliği, t zamanında sadece bir satır verinin gösterilmesini sağlamasıdır. Bu noktada GridView kontrolünden ayrılmasına rağmen, bir veri kümesi üzerinden navigasyon linkleri yardımıyla hareket edilmesincede izin verir. Konuyu daha iyi anlayabilmek için, bir önceki makalemizde yer alan Access veritabanımızı kullanacağımız bir örnek geliştireceğiz. Bu örneğimizde de, Access veri kaynağımıza bağlanmak ve Select, Insert, Update, Delete işlemlerini gerçekleştirebilmek amacıyla bir AccessDataSource bileşeni kullanacağız. Bu bileşenimizi aşağıdaki aspx kodları ile oluşturalım.

```
<asp:AccessDataSource ID="AccessDataSource1" Runat="server" DataFile="Data/veriler.mdb" SelectCommand="Select * From MailListesi" InsertCommand="Insert Into MailListesi (Ad,Soyad,Mail) Values (@Ad,@Soyad,@Mail)" UpdateCommand="Update MailListesi Set Ad=@Ad,Soyad=@Soyad,Mail=@Mail Where ID=@ID" DeleteCommand="Delete From MailListesi Where ID=@ID">
```

```
<InsertParameters>
```

```
<asp:Parameter Name="Ad"></asp:Parameter>
```

```
<asp:Parameter Name="Soyad"></asp:Parameter>
```

```
<asp:Parameter Name="Mail"></asp:Parameter>
```

```
</InsertParameters>
```

```
<UpdateParameters>
```

```
<asp:Parameter Name="Ad"></asp:Parameter>
```

```
<asp:Parameter Name="Soyad"></asp:Parameter>
```

```
<asp:Parameter Name="Mail"></asp:Parameter>
```

```
<asp:Parameter Name="ID"></asp:Parameter>
```

```
</UpdateParameters>
```

```
<DeleteParameters>
```

```
<asp:Parameter Name="ID"></asp:Parameter>
```

```
</DeleteParameters>
```

```
</asp:AccessDataSource>
```

AccessDataSource kontrolümüz, bağlanacağı veri bileşeni üzerinde, Select, Update, Insert ve Delete işlemlerinin gerçekleştirilebilmesini sağlamak amacıyla, SelectCommand, InsertCommand, DeleteCommand ve UpdateCommand özelliklerini kullanmaktadır. Özellikle veri girişi, güncelleme ve silme işlemleri için kullanılan sorgular, parametreler içermektedir. Bu parametreleri, hangi komut için kullanacak isek, o komuta ait parametre koleksiyonuna birer Parameter nesnesi olarak eklemeliyiz. Bu nedenle, InsertParameters, UpdateParameters ve DeleteParameters koleksiyonları kullanılmıştır. Bunun dışında, AccessDataSource kontrolümüzün, hangi access veritabanına bağlanacağını, DataFile özelliğine atadığımız dosya yol bilgisi ile belirliyoruz. Gelelim, DetailsView kontrolümüze. İlk aşamada bu kontrolümüzü aşağıdaki aspx kodları ile oluşturalım.

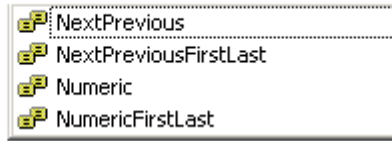
```

<asp:DetailsView ID="DetailsView1" Runat="server" DataSourceID="AccessDataSource1"
AllowPaging="True" PagerSettings-Mode="NextPreviousFirstLast" AutoGenerateInsertButton="True"
AutoGenerateEditButton="True" AutoGenerateDeleteButton="True" Font-Names="Verdana" Font-Size="X-Small"
DataKeyNames="ID">
</asp:DetailsView>

```

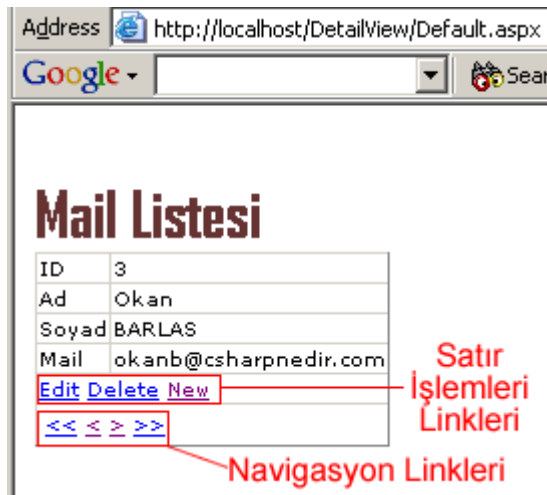
DetailsView kontrolümüzün, DataSourceID özelliğine atadığımız bileşen adı ile, hangi veri kaynağını ve sql sorgularını kullanacağını belirliyoruz. AllowPaging özelliği, her ne kadar GridView' daki sayfalama özelliğini çağırırsada burada sayfa numarası yerine, satır numaralarını temsil etmektedir. Yani n sayıda kaydı olan bir veri kümesi için, n adet sayfa numarası yer alacaktır. Çünkü DetailsView bileşeni, bir anda yalnız bir satırı ekranda göstermektedir. Bununla birlikte, sayfalama gösteriminin ne şekilde olacağını belirtmek amacıyla, PagerSettings-Mode özelliği kullanılmıştır. Burada verdiğimiz değer haricinde atayabileceğimiz diğer değerlerde sunlardır.

PagerSettings-Mode="



Şekil 1. PagerSettings-Mode değerleri.

AutoGenerateInsertButton özelliği bileşende satır ekleme için gerekli linkerlin gösterilmesini sağlar. Benzer şekilde, AutoGenerateEditButton özelliğide, düzenleme linklerinin gösterilmesini sağlar. Son olarak AutoGenerateDeleteButton özelliğide Delete linkinin gösterilmesini sağlamaktadır. Elbette bileşenimiz için kilit noktalardan biriside, DataKeyNames özelliğine atanan Field değeridir. Bu değer, önceki makalelerimizde hatırlayacağınız gibi, Update ve Delete sorguları (hatta bazı durumlarda Select sorgusu) içinde kullanılacak Primary Key alanını ifade etmektedir. Nitekim, tek bir satır üzerinde yapılacak güncelleme ve silme işlemleri, o satırın benzersiz bir numara ile ifade edilebilmesi halinde başarıya ulaşacaktır. İşte bu noktada primary key alanları büyük önem kazanır. Sayfamızı bu haliyle çalıştırdığımız takdirde aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 2. Uygulamanın çalışması sonucu.

Görüldüğü gibi, bir anda sayfada sadece tek bir satıra ait veriler görülmektedir. Veri kümesinin Select sorgusu ile elde edilen satırlar arasında gezebilmek için, Navigasyon linkeri kullanılır. Burada standart olarak bileşenimiz, Select sorgusuna göre tüm Field' ları getirmiştir. Ancak dilersek sadece belirli Field' ların kontrolümüzde görünmesini sağlayabiliriz. Nasıl ki, GridView kontrolünde bu iş için kullandığımız Columns koleksiyonu varsa, DetailsView bileşnindede aynı işi yapan Fields koleksiyonu vardır. Elbette koleksiyon isminin Fields olması tesadüfi değildir. Nitekim, DetailsView bileşeni, satırlarını Field bazında gösterir. Oysa GridView küme bazında ve sütunları esas alarak göstermektedir. Dolayısıyla bileşenimizin aspx kodlarını aşağıdaki gibi düzenleyerek sadece belirli field' ların görünmesini sağlayabiliriz.

```
<asp:DetailsView ID="DetailsView1" Runat="server" DataSourceID="AccessDataSource1"
AllowPaging="True" PagerSettings-Mode="NextPreviousFirstLast" AutoGenerateInsertButton="True"
AutoGenerateEditButton="True" AutoGenerateDeleteButton="True" Font-Names="Verdana" Font-Size="X-Small"
DataKeyNames="ID" AutoGenerateRows="false">
```

```
<Fields>
```

```
<asp:BoundField DataField="Ad" HeaderText="Adı"></asp:BoundField>
```

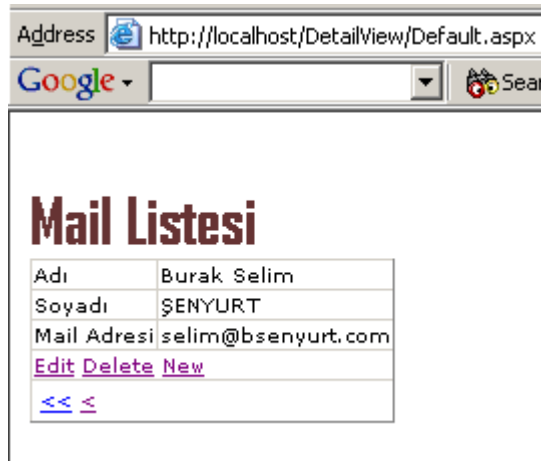
```
<asp:BoundField DataField="Soyad" HeaderText="Soyadı"></asp:BoundField>
```

```
<asp:BoundField DataField="Mail" HeaderText="Mail Adresi"></asp:BoundField>
```

```
</Fields>
```

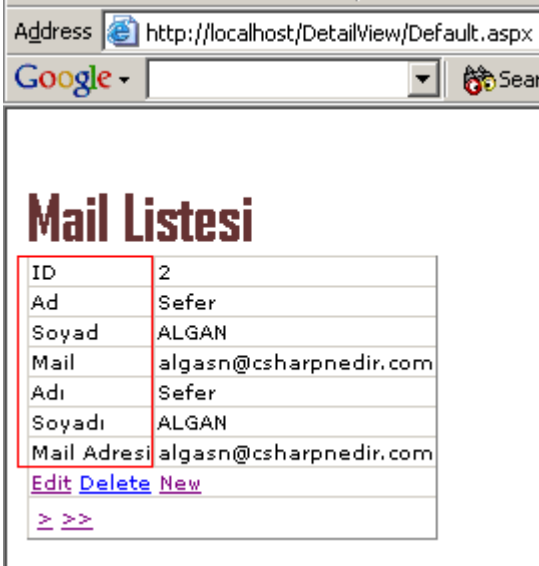
```
</asp:DetailsView>
```

Bu haliyle sayfamızı açtığımızda, aşağıdaki ekran görüntüsünü elde ederiz. Dikkat edecek olursanız sadece Ad,Soyad ve Mail alanları görülmektedir.



Şekil 3. Kendi belirlediğimiz alanların gösterilmesi.

Burada, AutoGenerateRows özelliğine dikkat etmenizi istiyorum. Bu özelliğe false değerini vermeseydik eğer, Select sorgusundan alınan tüm alanlar gösterilecek ve sonrada bizim belirlediğimiz alanlar ekrana gelecekti. Aşağıdaki şekilde görüldüğü gibi.



Şekil 4. AutoGenerateRows özelliğine false değeri vermeseydik.

Artık bileşenimiz yardımıyla, satır ekleme, silme ve güncelleme işlemlerini kolayca gerçekleştirebiliriz. Son olarak, sayfamızın kodlarını aşağıdaki hale getirerek sayfamızı ve en önemlisi DetalisView bileşenimizi biraz makyalayalım.

```
<%@ Page Language="C#" CompileWith="Default.aspx.cs" ClassName="Default_aspx" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:AccessDataSource ID="AccessDataSource1" Runat="server" DataFile="Data/veriler.mdb" SelectCommand="Select *
From MailListesi" InsertCommand="Insert Into MailListesi (Ad,Soyad,Mail) Values (@Ad,@Soyad,@Mail)"
UpdateCommand="Update MailListesi Set Ad=@Ad,Soyad=@Soyad,Mail=@Mail Where ID=@ID" DeleteCommand="Delete From
MailListesi Where ID=@ID">
        <InsertParameters>
          <asp:Parameter Name="Ad"></asp:Parameter>
          <asp:Parameter Name="Soyad"></asp:Parameter>
          <asp:Parameter Name="Mail"></asp:Parameter>
        </InsertParameters>
        <UpdateParameters>
          <asp:Parameter Name="Ad"></asp:Parameter>
          <asp:Parameter Name="Soyad"></asp:Parameter>
```

```

        <asp:Parameter Name="Mail"></asp:Parameter>
        <asp:Parameter Name="ID"></asp:Parameter>
    </UpdateParameters>
    <DeleteParameters>
        <asp:Parameter Name="ID"></asp:Parameter>
    </DeleteParameters>
</asp:AccessDataSource><br />
<span style="font-size: 24pt; color: #663333; font-family: Agency FB"><b>Mail Listesi<br />
</b></span>
<asp:DetailsView ID="DetailsView1" Runat="server" DataSourceID="AccessDataSource1"
AllowPaging="True" PagerSettings-Mode="NextPreviousFirstLast" AutoGenerateInsertButton="True"
AutoGenerateEditButton="True" AutoGenerateDeleteButton="True" Font-Names="Verdana" Font-Size="X-Small"
DataKeyNames="ID" CellPadding="4" BorderColor="#DEDFDE" GridLines="Vertical" BorderWidth="1px" ForeColor="Black"
BackColor="White" AutoGenerateRows="False" BorderStyle="None">
    <FooterStyle BackColor="#CCCC99"></FooterStyle>
    <RowStyle BackColor="#F7F7DE"></RowStyle>
    <PagerStyle ForeColor="Black" HorizontalAlign="Right" BackColor="#F7F7DE"></PagerStyle>
    <Fields>
        <asp:BoundField DataField="Ad" HeaderText="Adi"></asp:BoundField>
        <asp:BoundField DataField="Soyad" HeaderText="Soyadı"></asp:BoundField>
        <asp:BoundField DataField="Mail" HeaderText="Mail Adresi"></asp:BoundField>
    </Fields>
    <HeaderStyle ForeColor="White" Font-Bold="True" BackColor="#6B696B"></HeaderStyle>
    <EditRowStyle ForeColor="White" Font-Bold="True" BackColor="#CE5D5A"></EditRowStyle>
    <AlternatingRowStyle BackColor="White"></AlternatingRowStyle>
</asp:DetailsView>
</div>
</form>
</body>
</html>

```

Mail Listesi

Adı	Sefer
Soyadı	ALGAN
Mail Adresi	algasn@cssharpnedir.com
Edit Delete New	
> >>	

Mail Listesi

Adı	
Soyadı	
Mail Adresi	
Insert Cancel	

Şekil 5. DetailsView bileşenin makyajlanmış hali ve New linkine tıklandığında.

Örnek olarak, Yeni bir satır girdiğimizde, bu satırını başarılı bir şekilde veri kaynağına eklendiğini görürüz. Örneğin, aşağıdaki kayıt bilgisini Insert linki ile eklediğimizi düşünelim.

Mail Listesi

Adı	bili
Soyadı	geyts
Mail Adresi	bili@mikrosoft.com
Insert Cancel	

Şekil 6. Yeni satır verisi.

Insert linkine bastıktan sonra, tekrar son satıra gidersek, yeni satırımızın başarılı bir şekilde eklendiğini ve en önemlisi, Access tablosundaki otomatik olarak artan ID değerinin, başarılı bir şekilde oluşturulduğunu görürüz. Tekrar hatırlatmakta fayda var, burada yaptığımız veri girişi doğrudan, tabloya yansıtılacaktır.

Mail Listesi

Adı	bili
Soyadı	geyts
Mail Adresi	bili@mikrosoft.com
Edit Delete New	
<< >>	

Şekil 7. Eklenen satır.

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Asp.Net 2.0 ve Master Page Kavramı

Değerli Okurlarım Merhabalar,

Bu makalemizde, Master Pages kavramına giriş yapacak ve web uygulamalarının geliştirmesinde yaşamımıza getirdiği kolaylıkları incelemeye çalışacağız. İnternet sitelerini göz önüne aldığımızda, siteye ait sayfaların sıklıkla aynı şablonları kullandığını görürüz. Özellikle, header, footer, navigasyon ve advertisement alanları, çoğunlukla siteye ait tüm sayfalarda aynı yerlerde kullanılır. Bu, siteye ait sayfaların standart olarak aynı görünümde olmasını sağlamakla kalmaz, değişen içeriğinde ortak bir şablon üzerinde durmasına imkan tanır. Peki Asp.Net 2.0' ın bu kullanım için getirdiği yaklaşıma gelene kadar, sayfalarda ortak olarak kullanılan ve tasarımsal olarak sayfa koordinatlarında aynı yerlerde yer alan bu unsurlar hangi teknikler ile oluşturulmuştur?

Html' in ilk zamanlarında, bu tarz işlemleri gerçekleştirmek için, ortak olan alanlar kopyalanarak diğer sayfalara yapıştırılırdı. Yada, ana şablonu ihtiva eden bir sayfadan diğer sayfalar "save as" metodu ile oluşturulur ve içerikleri değiştirilirdi. Bu tekniğin en büyük

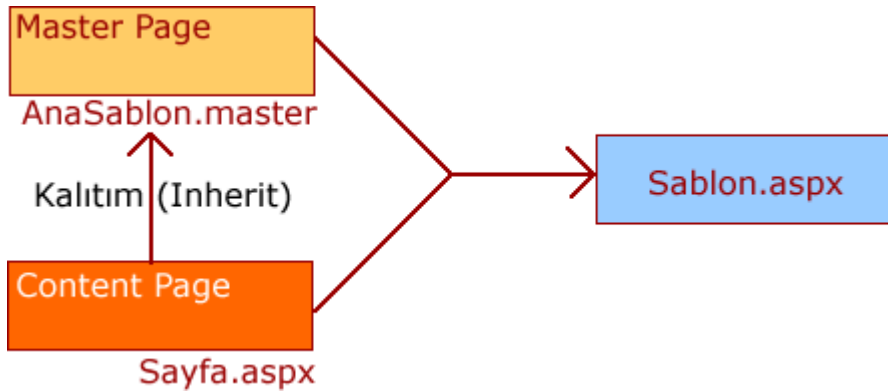
handikapı, şablondaki herhangi bir değişikliğin diğer sayfalara yansıtılması sırasında ortaya çıkmaktadır. Nitekim 100' lerce alt sayfaya aynı şablonu uygulamışsak bu gerçekten büyük bir problemdir.

Çözüm Asp ile gelmiştir. Asp, include takılarını kullanarak, sayfalarda tekrar eden içeriklerin kolayca kullanılabilmesini ve değişikliklerin tüm sayfalarda görünebilmesini sağlamıştır. Ancak elbetteki include takısının bir takım sorunları vardır. Bunlardan birisi, tasarım zamanında include takısının işaret ettiği içeriğin görünememesidir. Dolayısıyla sayfanın bütünün nasıl görüldüğünü inceleyebilmek için mutlaka çalıştırmak gerekmektedir. Diğer yandan, include tekniği takılar üzerine kurulu olduğundan, özellikle açık unutulmuş takılar sayfalarda istenmeyen Html çıktıların oluşmasına yol açmaktadır.

Asp.Net, bu tip ortak içeriklerin kullanılmasına daha güçlü ve etkin bir yaklaşımı getirmiştir. User Controls. Kullanıcı tanımlı kontroller, normal aspx içeriğine sahip olabilmekte ve .net mimarisinin güçlü özelliklerini kullanabilmektedir. Her ne kadar etkili bir teknik olsada, user control' ler içinde tek bir sorun öne çıkmaktadır. Tasarım zamanında, user control içeriğinin görülebilmesi.

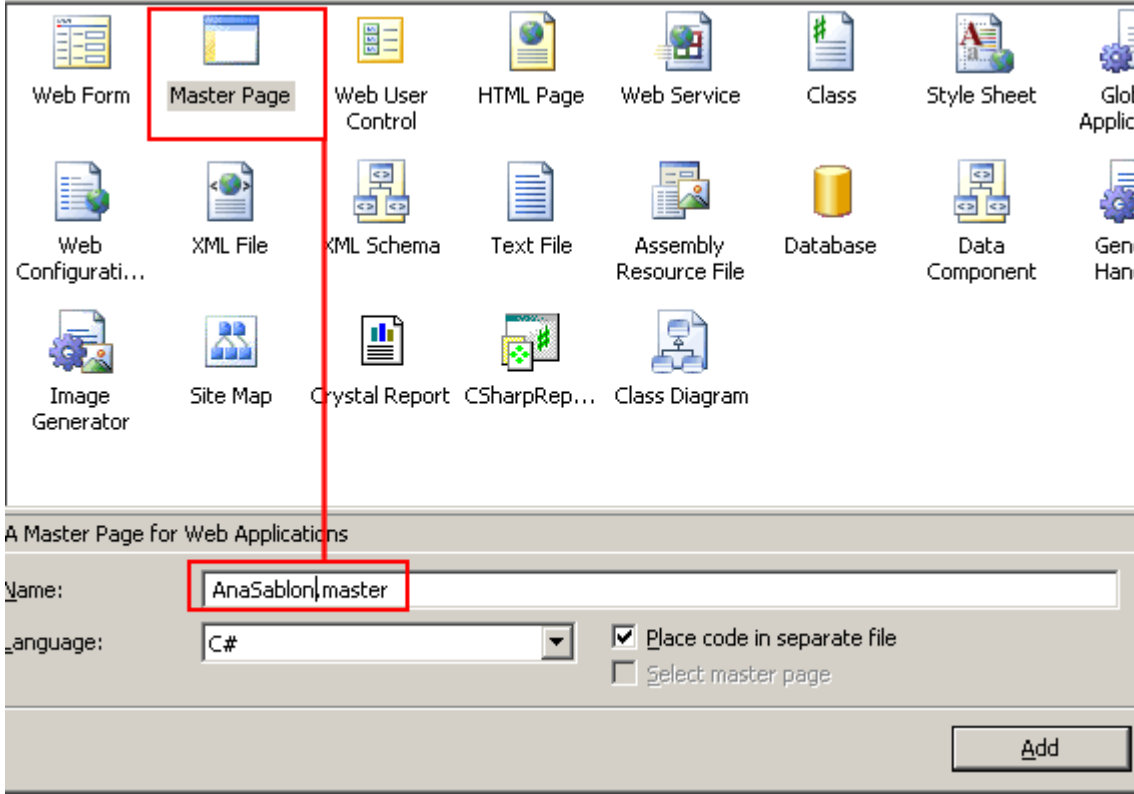
Asp.Net 2.0, Master Page yaklaşımı ile, yukarıda bahsedilen dezavantajları ortadan kaldırmayı başarmıştır. Bir Master Page, uygulandığı diğer aspx sayfalarının nasıl görünmesi gerektiğine karar veren bir şablona gibidir. Ancak, sağladığı ContentPlaceHolder bileşeni sayesinde, Master Page' leri uygulayan diğer aspx sayfalarının, istenilen içeriğe sahip olmasında sağlamaktadır. En güzel yanı ise, normal bir aspx sayfası gibi tasarlanabilmesi, yani html, image, server control gibi üyeleri içerebilmesidir. Bunlara ek olarak, olay güdümlü programlama modelinde destekler. Dolayısıyla bir Master Page aslında bir aspx sayfasından farksızdır.

Ancak asıl fark, bir Master Page bir aspx sayfasına uygulandığında ortaya çıkar. Master Page' i uygulayan bir aspx sayfası tarayıcıda açıldığında tarayıcıya gelen sayfa, Master Page ile aspx sayfasının birleştirilmesi sonucu ortaya çıkan başka bir aspx sayfasıdır. Bu, .net framework' ün getirdiği partial class tekniği sayesinde gerçekleştirilmektedir. Bunun, Master Page' i uygulayan aspx sayfalarına getirdiği değişik kodlama etkileride vardır.



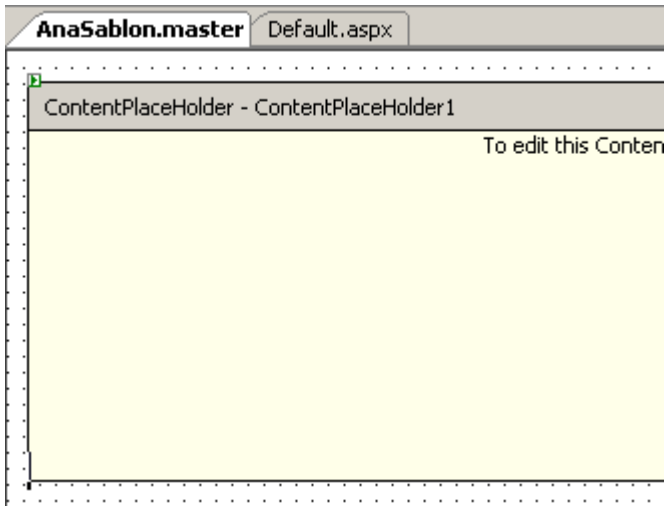
Şekil 1. Master Page ve aspx sayfalarının ortak çalışma mimarisi.

Bu kısa açıklamalardan sonra, Master Page' lerin ne olduğunu ve nasıl kullanıldıklarını anlamak amacıyla basit bir örnek geliştirelim. İlk olarak, Visual Studio.Net 2005' te bir web sitesi açalım. Sitemize Master Page eklemek için tek yapmamız gereken, Solution' ımıza sağ tıklamak ve Add New Item' den gelen pencerede, Master Page tipini seçmektir. Master Page' ler master uzantılı dosyalardır.



Şekil 2. Solution'a Master Page eklenmesi

Bu işlemin ardından Master Page' in standart olarak aşağıdaki gibi oluşturulduğunu görürüz.



Şekil 3. Varsayılan Master Page.

İşte burada ContentPlaceHolder1 bileşenimiz, bu Master Page' i uygulayacak olan sayfaların serbestçe erişebilecekleri ve içerik oluşturabilecekleri alanları tanımlamaktadır. Elbetteki bir Master Page' in bu şekilde olması beklenemez. Bu nedenle Master Page' imizi aşağıdaki ekran görüntüsünde olduğu gibi tasarlayabiliriz. Dikkat edecek olursanız, Master Page' lerde, normal aspx sayfaları gibi tasarlanabilirler. Bir başka deyişle, Html kodları, aspx bileşenleri vb. içerebilirler.



Şekil 4. Master Page sayfamızının tasarımı.

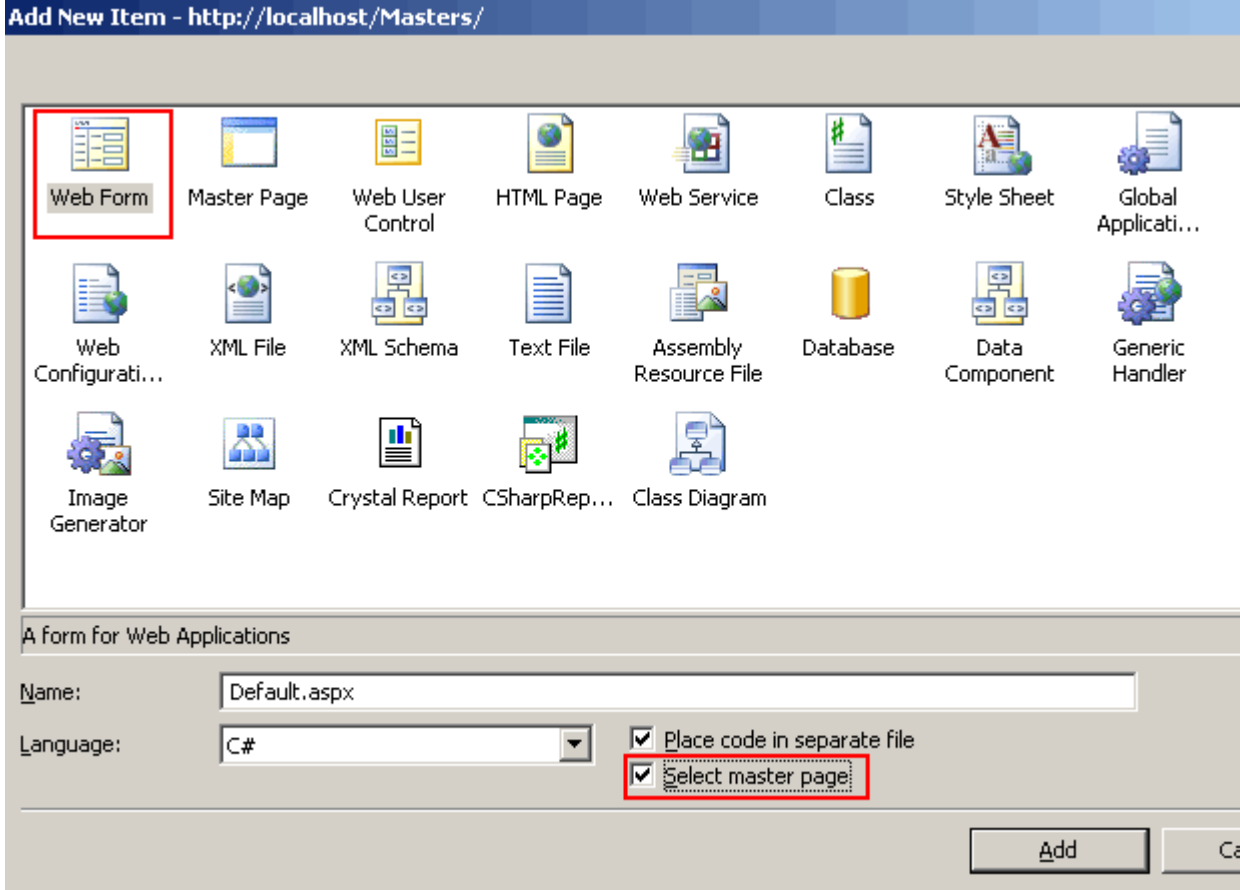
Burada standart olarak bir web sayfasının tasarlanmasından farklı bir işlem yapılmamıştır. En önemli nokta Master Page' i uygulayacak sayfaların içeriklerini yazabilecekleri ContentPlaceHolder bileşeninin kullanılmasıdır. Dilersek bir Master Page içinde, birden fazla ContentPlaceHolder bileşeninde yer verebiliriz. Master Page' in aspx kodlarına baktığımızda normal aspx sayfalarına göre en önemli değişik page direktifi yerine master direktifinin kullanılmasıdır. Master direktifi sayfanın bir Master Page olduğunu belirtmektedir.

```
<%@ Master Language="C#" CompileWith="AnaSablon.master.cs" ClassName="AnaSablon_master" %>
```

ContentPlaceHolder bileşeni ise,

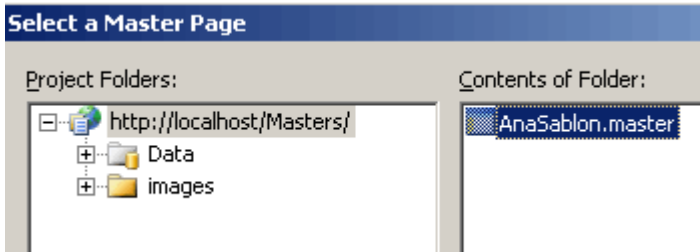
```
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" Runat="server"></asp:ContentPlaceHolder>
```

aspx kodları ile tanımlanır. Sayfanın dikkate değer bir özelliğide, form takılarını içermesidir. Nitekim biraz sonra göreceğimiz gibi bunun, Master Page' i uygulayan aspx sayfalarına etkisi vardır. Şimdi dilerseniz, yeni bir aspx sayfasına oluşturduğumuz Master Page' i nasıl uygulayacağımıza bir bakalım. Öncelikle, Add New Item iletişim kutusunu açalım ve dosya tipi olarak Web Form' u seçelim. Ardından, sayfamıza uygulamak istediğimiz Master Page' i seçebilmek amacıyla, Select Master Page kutucuğunu işaretleyelim.



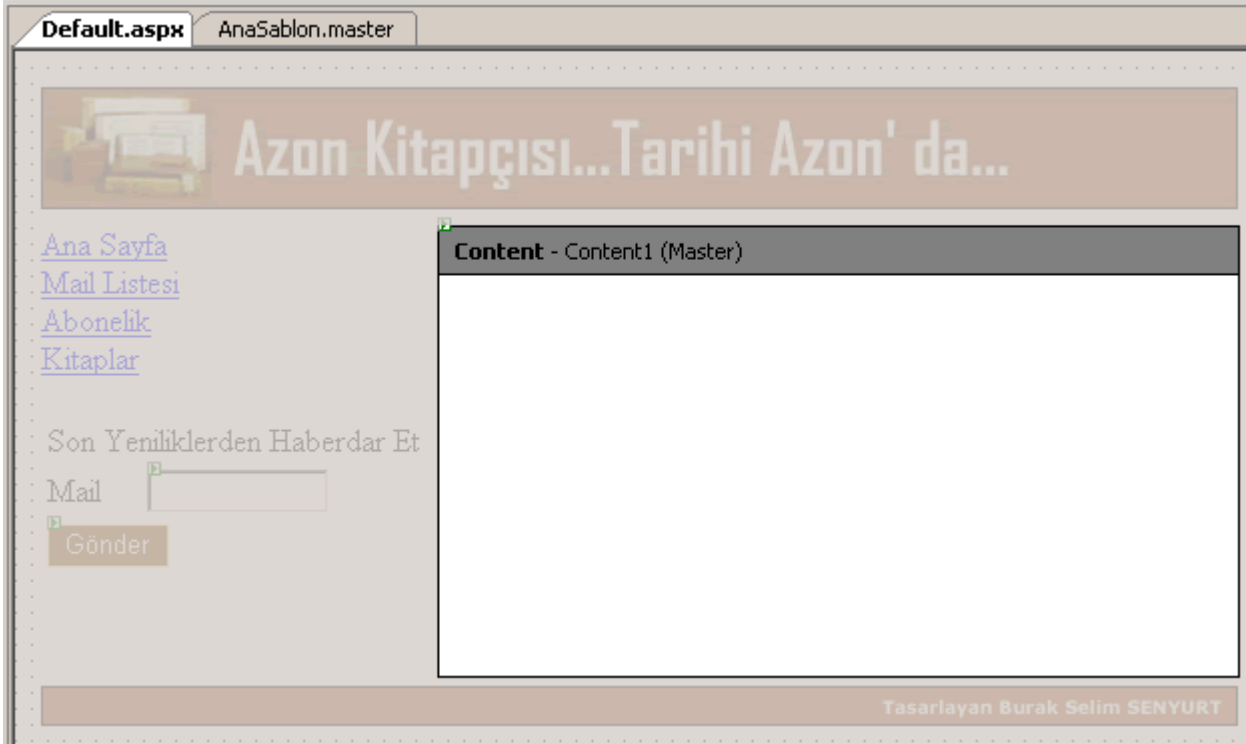
Şekil 5. Yeni bir web formun Master Page tabanlı oluşturulması.

Bu durumda Add buttonuna bastığımızda, sayfamıza uygulamak istediğimiz Master Page' i seçeceğimiz iletişim kutusu ekrana gelecektir.



Şekil 6. Master Page' in seçilmesi.

Bu adımda tamamladığımızda, default.aspx sayfamız aşağıdaki gibi oluşturulacaktır. Dikkat edecek olursanız, sadece Master Page' deki ContentPlaceHolder bileşeninin bulunduğu alan düzenlenebilir yapıdadır. Diğer kısımlar için düzenleme ve değiştirme gibi işlemleri gerçekleştirme imkanımız yoktur. Bu sayede web formunun, Master Page' in izin verdiği görünümde olması ve kendisine ayrılan alanda istediği içeriği oluşturmasına izni verilmiş olunur.



Şekil 7. Master Page' in bir web formuna uygulanması sonucu.

Bu noktada, web formumuzun aspx kodlarına bakacak olursak sadece tek bir satırın olduğunu görürüz.

```
<%@ Page Language="C#" MasterPageFile="~/AnaSablon.master" CompileWith="Default.aspx.cs" ClassName="Default_aspx"
Title="Untitled Page" %>
```

Bu tek satır aslında çok şey ifade etmektedir. Herşeyden önce, MasterPageFile özelliği, sayfaya uygulanan Master Page' in yolunu belirtir. Bu, formun bir Master Page' i uyguladığını başka bir deyişle master page' den türetilerek üretildiğini gösterir. Kendi sınıfı ve code-behind dosyası vardır.

Eğer sayfadaki Content alanı içerisinde düzenleme yapmak istersek bunu şu an için gerçekleştiremediğimizi görürüz. Bunu sağlayabilmek için, Master Page' de yer alan ContentPlaceHolder bileşeninin, bu sayfada bir Content bileşeni ile eşleştirilmesi gerekmektedir. Bunun için, web formumuza aşağıdaki aspx kodlarını yazmamız yeterli olacaktır. Content bileşeninin, ContentPlaceHolderID özelliği, uygulanan Master Page' deki hangi ContentPlaceHolder bileşenini eşleştireceğini belirtmektedir. Bu özelliğin değeri, birden fazla ContentPlaceHolder' ın, Master Page' i uygulayan sayfalarda eşleştirilmesinde önem kazanır.

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
</asp:Content>
```

Artık, içeriği istediğimiz gibi değiştirebiliriz. Sunucu elemanları, html kodları, resimler vesaire. Örneğin, bir Access tablosundan buradaki Content alanı içindeki bir GridView bileşenini dolduralım.

Kitap.aspx	AnaSablon2.master	Default.aspx	AnaSablon.master



Azon Kitapçısı...Tarihi Azon' da...

[Ana Sayfa](#)
[Mail Listesi](#)
[Kitaplar](#)
[Abonelik](#)

ContentPlaceHolder - ContentPlaceHolder1

To edit this ContentPlaceHolder control, click here and type text or drag a control from the Toolbox.

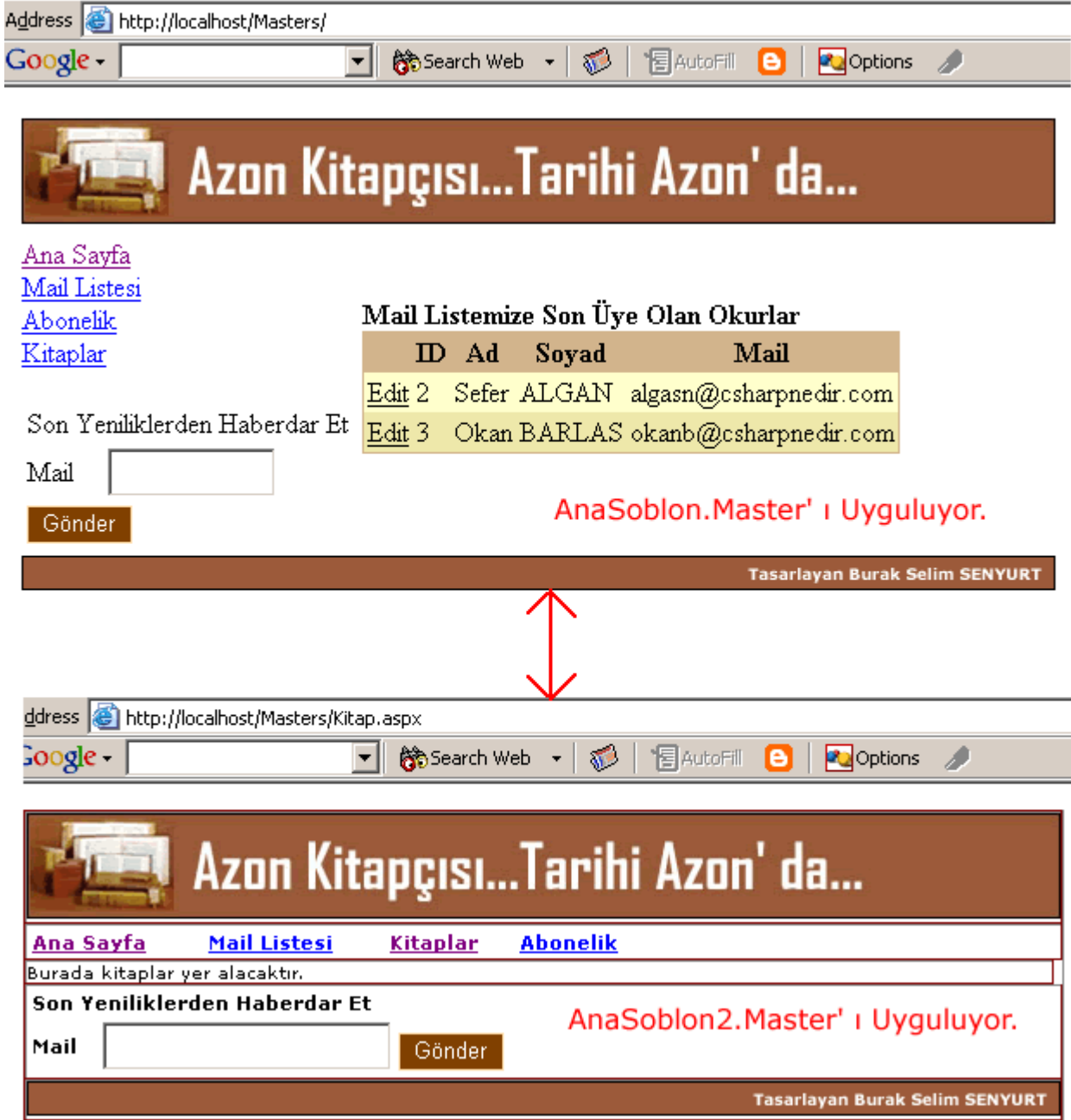
Son Yeniliklerden Haberdar Et

Mail

Tasarlayan Burak Selim SENYURT

Şekil 10. AnaSablon2.Master

Bu Master Page' i başka bir sayfaya kolayca uygulayabiliriz. Örneğin Kitap.aspx isimli bir web formu oluşturalım ve formumuzu AnaSablon2.master' dan türetelim. Böylece, uygulamamız içerisinde iki farklı Master Page yapısını kullanan sayfalara izin vermiş oluyoruz.



Şekil 11. Bir uygulamada farklı Master Page' lerin kullanılması.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde kısaca, Master Page' lerin, uygulandıkları aspx sayfaları ile kombine bir şekilde nasıl çalıştıklarını incelemeye çalıştık. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

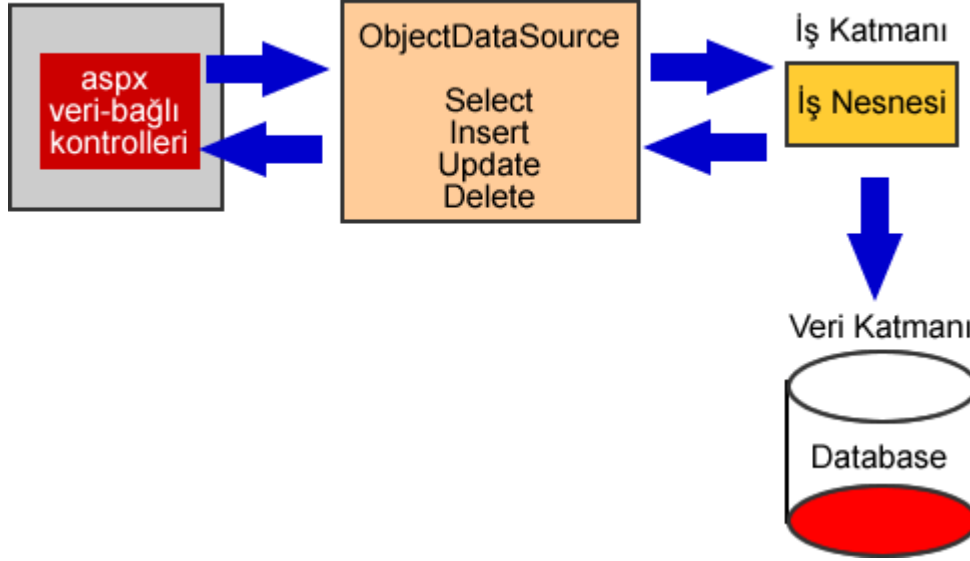
Asp.Net 2.0 ve ObjectDataSource Kontrolü

Değerli Okurlarım Merhabalar,

Bu makalemizde ObjectDataSource bileşenini incelemeye çalışacağız. ObjectDataSource bileşeni Asp.Net 2.0 ile gelen yeni bileşenlerden birisidir. Görevi, 3 katlı mimarinin uygulanması halinde, verikaynağı ile sunum katmanında yer alan veri bağlı

kontroller arasındaki veri alışverişinin, iş katmanı üzerindeki herhangi bir nesne yardımıyla gerçekleştirilebilmesini sağlamaktır. Başka bir deyişle, veri-bağlı kontroller ile, iş katmanında yer alan iş nesnesi arasındaki iletişimi sağlamaktadır.

Sunu Katmanı



Şekil 1. ObjectDataSource bileşenin 3 katlı mimarideki rolü.

Örneğimizi incelediğimizde ObjectDataSource nesnesinin 3 katlı mimarideki yerini daha kolay anlama fırsatı bulacağız. Şimdi şu senaryoyu göz önüne alalım. Web sitemizdeki veri bağlı kontrollerin herhangi bir veri kaynağından veri çekmek, veri eklemek, veri silmek ve veri güncellemek gibi işlemler için bir sınıf ve metodlarını kullandığını düşünelim. İşte bu sınıfın sağlamış olduğu fonksiyonellikleri, sunum katmanında yer alan veri-bağlı kontroller ile kolayca gerçekleştirebilmek amacıyla ObjectDataSource nesnesini kullanabiliriz. Örnek olarak, Personel ile ilgili kayıtları tutan bir access tablomuz olduğunu düşünelim. Amacımız ilk olarak, bu tablodaki verileri alıp getirecek ve yeni bir satır veri girilmesini sağlayacak bir metoda sahip olan bir sınıf yazmak olsun. Bunun için aşağıdaki kodlara sahip bir sınıf geliştirelim.

```

using System;
using System.Data;
using System.Data.OleDb;
using System.Web;

public class IsNesnesi
{
    public DataSet VerileriAl()
    {
        OleDbConnection con = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;data source=" +
        HttpContext.Current.Server.MapPath("~/Data/Veriler.mdb"));
        OleDbDataAdapter da = new OleDbDataAdapter("Select * From Personel", con);
        DataSet ds = new DataSet();
    }
}
  
```

```

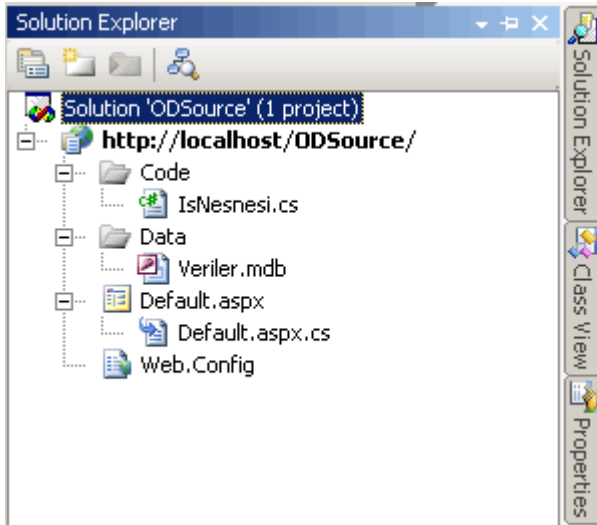
        da.Fill(ds);
        return ds;
    }

    public int VeriGir(string ad, string soyad, string mail, string departman, DateTime giris)
    {
        OleDbConnection con = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;data source=" +
HttpContext.Current.Server.MapPath("~/Data/Veriler.mdb"));
        OleDbCommand cmd = new OleDbCommand("Insert Into Personel (PerAd,PerSoyad,Departman,Mail,GirisTarihi) Values ('" +
ad + "','" + soyad + "','" + departman + "','" + mail + "','" + giris + "')", con);
        con.Open();
        int sonuc = cmd.ExecuteNonQuery();
        con.Close();
        return sonuc;
    }

    public IsNesnesi()
    {
    }
}

```

VerileriAl isimli metodumuz, Veriler isimli Access veritabanına bağlanarak burada yer alan Personel isimli tablodan tüm satırları alıyor ve bu satırları yüklediği DataSet nesnesini geri döndürüyor. VeriGir isimli metodumuz ise, parametreler alarak, tablomuza yeni bir satır ekliyor ve OleDbCommand nesnesinin ExecuteNonQuery metodunun çalıştırılması sonucu dönen integer değeri çağırıldığı yere aktarıyor. Şimdi bu sınıfımızı kullanacak bir web uygulaması inşa edelim. Web sitemizde, iş nesnemize ait IsNesnesi sınıfını Code klasörümüzün altına yerleştireceğiz. Bununla birlikte, veritabanımızda yine Data klasörü altına kopyalayalım.



Şekil 2. Uygulama yapımız.

Bu ön hazırlıklardan sonra, default.aspx sayfamızda aşağıdaki kontrolleri barındıracak şekilde oluşturalım.

Default.aspx.cs Code/IsNesnesi.cs **Default.aspx**

ObjectDataSource - ObjectDataSource1

Personel Adı	Personel Soyadı	Departmanı	İşe Başlama Tarihi	Mail Adresi
Databound	Databound	Databound	Databound	Databound
Databound	Databound	Databound	Databound	Databound
Databound	Databound	Databound	Databound	Databound
Databound	Databound	Databound	Databound	Databound
Databound	Databound	Databound	Databound	Databound

Yeni Eleman Girişi

Ad

Soyad

Departman

Giriş Tarihi

Mail

Şekil 3. default.aspx sayfamızın yapısı.

Burada iş nesnesi ile formda yer alan bağlı kontroller arasındaki ilişkiyi sağlamak üzere bir ObjectDataSource nesnemiz yer almaktadır. İlk önce, ObjectDataSource bileşenimizin aspx kodlarını aşağıdaki gibi tamamlayalım.

```
<asp:ObjectDataSource ID="ObjectDataSource1" Runat="server" TypeName="IsNesnesi"
SelectMethod="VerileriAl" InsertMethod="VeriGir">
  <InsertParameters>
    <asp:Parameter Name="ad"></asp:Parameter>
    <asp:Parameter Name="soyad"></asp:Parameter>
    <asp:Parameter Name="mail"></asp:Parameter>
    <asp:Parameter Name="departman"></asp:Parameter>
    <asp:Parameter Name="giris" Type="DateTime"></asp:Parameter>
  </InsertParameters>
</asp:ObjectDataSource>
```

ObjectDataSource bileşenimizin başarılı bir şekilde işleyebilmesi için en önemli gereklilikler, select, insert, update ve delete işlemleri için iş nesnesi üzerindeki hangi metodların kullanacağını bildirilmesidir. Biz örneğimizde ye alan iş nesnesine ait sınıf içerisinde, veri çekmek ve veri girişi için iki metod tanımladık. ObjectDataSource bileşeninin SelectMethod ve InsertMethod özelliklerine, ilgili Select ve Insert işlevlerini gerçekleştiren iş nesnesi metodlarının adlarını aktarıyoruz. Artık ObjectDataSource ile veriye bağlanacak

olan nesneler, veri çekmek ve veri girişi için burada belirtilen metodları kullanacaklardır. Tabiki, ObjectDataSource' un bu bilgiler dışında ilgili metodları içeren sınıf hakkında da bilgiye sahip olması gerekir. İşte bunun içinde, TypeName özelliğine ilgili iş nesnesinin sınıf adı atanır.

Şunu hemen belirtelim ki, burada tanımladığımız sınıf başka bir isim alanı altında olabilir. Bu durumda, bu sınıfa tam olarak ulaşılacak tip bilgisinin TypeName özelliğinde belirtilmesi gerekir. Örneğin, sınıfımız IsKatmani isimli bir isim alanında bulunuyorsa bu durumda TypeName özelliğine IsKatmani.IsNesnesi değeri girilmelidir.

Diğer DataSource bileşenlerinde olduğu gibi, Insert, Update ve Delete gibi işlemler parametreler yolu ile gerçekleştirilmektedir. Bizimde burada, Insert metodumuzda kullandığımız bazı parametreler mevcuttur. İşte bu parametreleri temsil etmesi için, ObjectDataSource kontrolünün InsertParameters koleksiyonu kullanılmıştır. Delete ve Update işlemlerinde de DeleteParameters ve UpdateParameters koleksiyonları kullanılır.

Gelelim, formumuzda yer alan veri bağlı kontrollere. Veri çekme işlemi sonrasında, tablomuzda yer alan satırları GridView kontrolünde göstermek istediğimiz için, GridView kontrolüne ait aspx kodlarını aşağıdaki gibi yazmalıyız.

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="ObjectDataSource1" AutoGenerateColumns="false">
  <Columns>
    <asp:BoundField DataField="PerAd" HeaderText="Personel Adı"></asp:BoundField>
    <asp:BoundField DataField="PerSoyad" HeaderText="Personel Soyadı"></asp:BoundField>
    <asp:BoundField DataField="Departman" HeaderText="Departmanı"></asp:BoundField>
    <asp:BoundField DataField="GirisTarihi" DataFormatString="{0:dd.mm.yyyy}" HeaderText="İşe Başlama Tarihi"></asp:BoundField>
    <asp:BoundField DataField="Mail" HeaderText="Mail Adresi"></asp:BoundField>
  </Columns>
</asp:GridView>
```

GridView kontrolümüzde DataSourceID özelliğine ObjectDataSource bileşenimizin ID değerini atamak ile, çalışma zamanında GridView kontrolünde, ObjectDataSource' un iş nesnesi üzerinden çalıştırdığı Select metodundan dönen DataSet' e ait satırların gösterilmesini sağlamış oluyoruz. Eğer sayfamızı bu haliyle çalıştıracak olursak aşağıdaki ekran görüntüsünü elde ederiz.

Address  http://localhost/ODSource/

Google  Search Web  0 blocked  AutoFill  Options

Personel Adı	Personel Soyadı	Departmanı	İşe Başlama Tarihi	Mail Adresi
Burak Selim	Senyurt	Yazılım	01.00.2000	selim@bsenyurt.com
Sefer	Algan	Yazılım	01.00.2000	algans@csharpnedir.com
Okan	Barlas	Yazılım	01.00.2000	okanb@csharpnedir.com

Yeni Eleman Girişi

Ad

Soyad

Departman

Giriş Tarihi

Mail

Ekle

Şekil 4. Select metodunun çalışması sonucu.

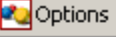
Gelelim Insert işlevinin nasıl gerçekleştirileceğine. Yeni bir satır eklemek için Ekle başlıklı button kontrolümüze bir kaç satır kod yazacağız.

```
void btnEkle_Click(object sender, EventArgs e)
{
    ObjectDataSource1.InsertParameters["ad"].DefaultValue = txtAd.Text;
    ObjectDataSource1.InsertParameters["soyad"].DefaultValue = txtSoyad.Text;
    ObjectDataSource1.InsertParameters["mail"].DefaultValue = txtMail.Text;
    ObjectDataSource1.InsertParameters["departman"].DefaultValue = txtDepartman.Text;
    ObjectDataSource1.InsertParameters["giris"].DefaultValue = txtGiris.Text;
    ObjectDataSource1.Insert();
}
```

ObjectDataSource kontrolümüz, iş nesnesi üzerindeki Insert işlevini gerçekleştirecek metodu bilmektedir. Ayrıca, bu metoda göndermesi gereken parametrelerin ne olacağını bilir. Bununla birlikte, Insert işlevini gerçekleştirecek olan iş nesnesi metodu için gerekli parametre değerlerinin bir şekilde gönderilmesi gerekmektedir. Bu amaçla, ObjectDataSource nesnesinin InsertParameters koleksiyonundaki her bir parametreye gerekli değerler gönderilir. Daha sonra ise bu parametrelerin değerleri, iş nesnesindeki metod içinde var olan karşılıklarına gönderilir. Bu gönderme emrini ise Insert metodu verir. Başka bir deyişle, Insert metodu, ObjectDataSource kontrolünün InsertParameters koleksiyonunda yer alan parametre değerlerini, iş nesnesi üzerindeki ilgili metoda gönderir ve bu metodu çalıştırır. Tabi burada ObjectDataSource bileşenindeki InsertParameters koleksiyonunda yer alan

parametrelerinin Name özelliklerinin değerlerinin, iş nesnesindeki VeriGir metodundaki parametre isimleri ile aynı olduğunda dikkat etmemiz gerekmektedir. Sonuç olarak veri giriş işlemi gerçekleşmiş olur. Şimdi uygulamamızı denersek, yeni satırlar ekleyebildiğimizi görürüz.

Address  http://localhost/ODSource/Default.aspx

Google    0 blocked  

Personel Adı	Personel Soyadı	Departmanı	İşe Başlama Tarihi	Mail Adresi
Burak Selim	Senyurt	Yazılım	01.00.2000	selim@bsenyurt.com
Sefer	Algan	Yazılım	01.00.2000	algans@csharpnedir.com
Okan	Barlas	Yazılım	01.00.2000	okanb@csharpnedir.com

Yeni Eleman Girişi	
Ad	Rio
Soyad	De Jeneryo
Departman	Yazılım
Giriş Tarihi	01.05.1986
Mail	rio@r.com
Ekle	

Şekil 5. Yeni satırı eklemekten önce.

Address <http://localhost/ODSource/Default.aspx>

Google Search Web 0 blocked AutoFill Options

Personel Adı	Personel Soyadı	Departmanı	İşe Başlama Tarihi	Mail Adresi
Burak Selim	Senyurt	Yazılım	01.00.2000	selim@bsenyurt.com
Sefer	Algan	Yazılım	01.00.2000	algans@csharpnedir.com
Okan	Barlas	Yazılım	01.00.2000	okanb@csharpnedir.com
Rio	De Jeneryo	Yazılım	01.00.1986	rio@r.com

Yeni Eleman Girişi

Ad	Rio
Soyad	De Jeneryo
Departman	Yazılım
Giriş Tarihi	01.05.1986
Mail	rio@r.com
Ekle	

Şekil 6. Yeni satır girildikten sonra.

Insert işleminde dikkat ederseniz, TextBox kontrollerine girdiğimiz değerleri InsertParameters koleksiyonundaki ilgili parametrelere aktardık. Dilersek, parametre değerlerinin aktarımını ilgili kontroller üzerinden doğrudanda gerçekleştirebiliriz. Bunun için, InsertParameters koleksiyonunda Parameter nesneleri yerine, ilgili parametre değerini taşıyan kontrolü temsil edecek ControlParameter nesneleri kullanılır.

```
<asp:ObjectDataSource ID="ObjectDataSource1" Runat="server" TypeName="IsNesnesi"
SelectMethod="VerileriAl" InsertMethod="VeriGir">
  <InsertParameters>
    <asp:ControlParameter ControlID="txtAd" PropertyName="Text" Name="ad"></asp:ControlParameter>
    <asp:ControlParameter ControlID="txtSoyad" PropertyName="Text" Name="soyad"></asp:ControlParameter>
    <asp:ControlParameter ControlID="txtMail" PropertyName="Text" Name="mail"></asp:ControlParameter>
    <asp:ControlParameter ControlID="txtDepartman" PropertyName="Text" Name="departman"></asp:ControlParameter>
    <asp:ControlParameter ControlID="txtGiris" PropertyName="Text" Type="DateTime"
Name="giris"></asp:ControlParameter>
  </InsertParameters>
</asp:ObjectDataSource>
```

Dikkat edecek olursanız, ControlID ile hangi kontrolün kullanacağı, PropertyName özelliği ile de bu kontrolün hangi özelliğinin değerinin alınacağı belirtilir. Name özelliği ise, iş nesnesindeki ilgili insert metodunda kullanılacak parametreyi işaret etmektedir ve

iş nesnesindeki metod parametresindeki isim ile aynıdır. Bu durumda, insert işlevini gerçekleştirebilmek için tek yapmamız gereken ObjectDataSource bileşenine ait Insert metodunu çağırmak olacaktır. Insert metodu çalıştırıldığında ObjectDataSource bileşeni, InsertParameters koleksiyonunda belirtilen kontroller üzerindeki değerleri alıp ilgili insert metoduna göndermektedir.

```
void btnEkle_Click(object sender, EventArgs e)
{
    ObjectDataSource1.Insert();
}
```

Delete ve Update işlevleride Insert işlevine benzer şekilde gerçekleştirilir. Bu kez, ObjectDataSource bileşeni için DeleteCommand ve UpdateCommand özellikleri bildirilir ve ilgili parametre koleksiyonları eklenir. Bu makalemizde kısaca ObjectDataSource bileşenini incelemeye çalıştık. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Ado.Net 2.0 ve Bulk-Data Kopyalama Mekanizması

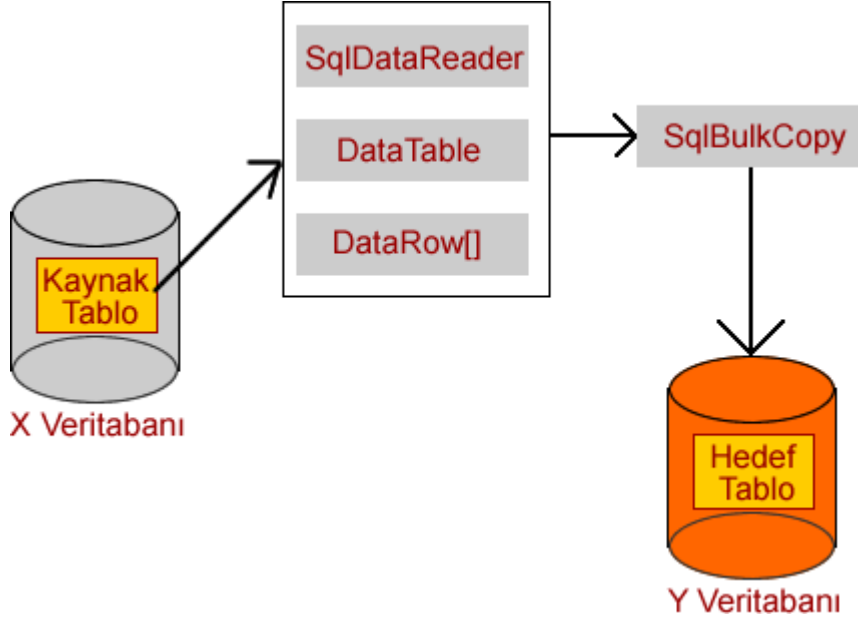
Değerli Okurlarım Merhabalar,

Sql Server'da bir veritabanı tablosundan, başka bir hedef tabloya veri taşıma işlemi bulk-data kopyalama olarak adlandırılır. Veritabanı yöneticileri çoğunlukla bu operasyonu gerçekleştirmek amacıyla, BCP adı verilen komut satırı aracını kullanırlar. Burada amaç, kaynak tablodaki satırların veya bir satır kümesinin farklı konumda olabilecek bir tabloya taşınmasıdır. Hedef tablo aynı veritabanında olabileceği gibi, diğer bir sql sunucusu üzerindeki başka bir veritabanında da yer alabilir. Ado.Net 2.0' da SqlClient isim alanına eklenen yeni sınıflar yardımıyla bu işlemleri yönetimli kodda (managed-code) gerçekleştirme imkanına da artık sahibiz. Bu makalemizde, bu işlemleri gerçekleştirmek için kullanabileceğimiz yeni Ado.Net 2.0 sınıflarını incelemeye çalışacağız.

Bulk-Data kopyalama işlemi için Ado.Net 2.0 ile gelen en önemli sınıf, SqlClient isim alanında yer alan SqlBulkCopy sınıfıdır. Bu sınıfa ait nesne örnekleri yardımıyla, kaynak tablodan hedef tabloya veri transferi işlemleri kolayca gerçekleştirilebilir. Bu işlemler sırasında SqlBulkCopy nesne örnekleri, taşıma işlemi varsayılan olarak açtığı bir transaction içerisinde gerçekleştirilmektedir. Yani, hedef tabloya yapılan taşıma işlemleri sırasında oluşabilecek olan hatalar sonrasında, transaction işlemi iptal edilerek roll-back operasyonu gerçekleşir ve hedef tabloya o ana kadar girilen satırlar geri alınır.

Burada önemli olan noktalardan birisi, kaynak ve hedef tabloların aynı şema yapısına sahip olmalarının önemli olmayışındır. Yani, aynı alan adları, eşit alan sayıları ve aynı alan sıraları olmak zorunda değildir. Nitekim, hedef tablo ile kaynak tablo arasında alan eşleşmelerinde uyumsuzluk olabilir. Örneğin, alan adları ve sayılar birbirinden farklı olabilir. İşte bu durumda, kaynak ve hedef tablodaki alanları birbirleriyle eşleştirmekte kullanılan SqlBulkCopyColumnMapping sınıfına ait nesne örnekleri kullanılır. Bu sınıf yardımıyla kaynak ve hedef alanların kolayca eşleştirilmesi sağlanmış olur. Bulk-Data kopyalama işlemi için önemli olan bir diğer sınıf ise, SqlBulkCopyColumnMappingCollection' dir. Bu sınıf ise, tahmin edeceğimiz gibi SqlBulkCopyColumnMapping sınıfı türünden nesne örneklerinin bir koleksiyonunu ifade etmektedir.

Bulk-Data kopyalama operasyonunda, kaynak veriler için yine SqlClient isim alanındaki standart nesneler kullanılabilir. Esasen bu tip bir operasyonda, kaynak verileri herhangi bir SqlDataReader nesnesinden, bir DataTable' dan veya bir DataRow dizisinden alabiliriz. Hatta bir Xml dökümanını da kaynak olarak kullanabiliriz. Böylece yönetimsel koda kazandırılan bu imkanlar ile veri taşıma işlemi için büyük esneklik kazanmış olmaktadır.



Şekil 1. *SqlBulkCopy Sınıfının Çalışma Şekli.*

Bulk-Data kopylama işleminin daha kolay anlaşılabilmesi için elbette örnekler ile olayı incelememiz daha faydalı olacaktır. Şimdi aşağıdaki basit Console uygulamasını göz önüne alalım.

```
#region Using directives
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;
```

```
#endregion
```

```
namespace BulkCopy
```

```
{
```

```
    class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated security=SSPI");
```

```
            /*Hedef tablodaki satırları önlem olarak siliyoruz.*/
```

```
SqlCommand cmd = new SqlCommand("DELETE * FROM YEDEK_MailList", con);
con.Open();
cmd.ExecuteNonQuery();

/* Kaynak tablodan verileri SqlDataReader yardımıyla çekiyoruz.*/
SqlCommand cmdCek = new SqlCommand("SELECT * FROM MailList", con);
SqlDataReader dr;
dr = cmdCek.ExecuteReader();

/*SqlBulkCopy nesnemizi hedef bağlantıyı belirterek oluşturuyoruz.*/
SqlBulkCopy bc = new SqlBulkCopy(con);

/* Hedef tabloyu belirtiyoruz.*/
bc.DestinationTableName = "YEDEK_MailList";

/*WriteToServer metodu ile SqlDataReader'dan okuduğumuz satırları, hedefe insert ediyoruz.*/
bc.WriteToServer(dr);

/* Nesneleri kapatıyoruz.*/
bc.Close();
dr.Close();
con.Close();
}
}
}
```

Örneğimizi incelemeden önce, Yukon sunucumuz üzerinde, AdventureWorks veritabanı altında MailList ve YEDEK_MailList isimli iki tablomuz olduğunu ve bu tabloların şema yapılarının birbirleriyle tamamen aynı olduklarını belirtelim.

dbo.YEDEK_Ma...ventureWorks)		dbo.MailList: ...AdventureWorks)	
	Column Name	Data Type	Allow Nulls
?	ID	numeric	☐
	AD	nchar	☐
	SOYAD	nchar	☐
▶	MAIL	nchar	☒
			☐

dbo.YEDEK_Ma...ventureWorks)		dbo.MailList: ...AdventureWorks)	
	Column Name	Data Type	Allow Nulls
▶?	ID	numeric	☐
	AD	nchar	☐
	SOYAD	nchar	☐
	MAIL	nchar	☒
			☐

Şekil 2. Her iki tablonda şema yapısı aynıdır.

Burada gerçekleştirdiğimiz operasyon, bulk-data kopyalama işleminin en basit halidir. Şimdi neler yaptığımıza biraz daha yakından bakalım. İlk olarak Yukon sunucumuzdaki AdventureWorks veritabanına bir bağlantı açtık. Daha sonra, tedbir olarak YEDEK_MailList tablosundaki tüm satırları sildik. Nitekim böyle bir işlem yapmasaydık YEDEK_MailList tablosuna taşınan satırlar sürekli arka arkaya eklenecekti. Daha sonra ise, klasik olarak kaynak tablomuzdan verileri bir SqlDataReader nesnesi yardımıyla çektik. Bizi asıl ilgilendiren kısım aşağıdaki kod satırlarının yer aldığı kısımdır.

```
SqlBulkCopy bc = new SqlBulkCopy(con);
```

```
bc.DestinationTableName = "YEDEK_MailList";
```

```
bc.WriteToServer(dr);
```

Burada ilk satırda, SqlBulkCopy nesnemizi o anki geçerli SqlConnection nesnesi ile oluşturuyoruz. Yapıcı metoda ait parametre hedef bağlantıyı temsil etmektedir. Nitekim, hedef tablomuz aynı sunucuda olmayabilir. Ya da, aynı veritabanı üzerinde olmayabilir. Bu nedenle burada bağlantıyı dikkatli seçmek gerekmektedir. Sonraki satırda ise, taşıma işleminin hedef alındığı tablo, SqlBulkCopy nesnemize DestinationTableName özelliği ile bildirilir. Buradaki tablo, SqlBulkCopy nesnesi örneklendirilirken parametre olarak verdiğimiz SqlConnection nesnesinin belirttiği bağlantı üzerinde aranır. Son satırımızda ise, WriteToServer metodu kullanılmıştır. Bu metoda da parametre olarak, kaynak tabloya ait veri kümesini taşıyan SqlDataReader nesnesi verilmiştir. Böylece WriteToServer metodu, SqlDataReader nesnesi ile MailList tablosundan okuduğumuz satırları, SqlBulkCopy nesnesi oluşturulurken belirtilen bağlantı üzerindeki hedef tabloya yazmaktadır.

Örneği çalıştırdığımızda, kaynak verilerin hedef tabloya taşındığını görürüz. Burada enteresan bir nokta vardır. Bu işlemi üst üste bir kaç kez uyguladığımızda, hedef tablodaki ID alanının değerlerinin sürekli olarak arttığını görürüz. Öyleki kaynak tablomuzda 1,2,3 olarak giden ID alanı değerleri, uygulama bir kaç kez çalıştırdıktan sonra hedef tabloda aşağıdaki gibi olabilir.

dbo.YEDEK_Ma...ventureWorks)		dbo.MailList: ...AdventureWorks)		dbo.Mail
ID	AD	SOYAD	MAIL	
19	Burak Selim	... ŞENYURT	... selim@bsenyurt....	
20	Sefer	... ALGAN	... algans@csharpn...	
21	Okan	... BARLAS	... okanb@csharpn...	
*				

Şekil 3. ID alanının durumu.

Her ne kadar biz ID alanını her iki tabloda otomatik olarak artan identity değeri ile belirtsek, Bulk-Data kopyalama işleminde, ID değerlerinin bozulmadan hedef tabloya yansıtılabilmeside mümkündür. Bunun için, SqlBulkCopy nesnemizin bir diğer yapıcı metodunu kullanırız.

```
public SqlBulkCopy(connectionString, SqlBulkCopyOptions copyOptions);
```

Bu yapıcı metod SqlBulkCopyOptions numaralandırıcısı türünden bir parametre daha alır. Bu numaralandırıcının alabileceği değerler şunlardır.

SqlBulkCopyOptions Numaralandırıcı Değeri	Açıklaması
CheckConstraints	Hedef tablodaki kısıtlamalar var ise bunlar göz önüne alınarak veri girişi gerçekleşir.
Default	Varsayılan değerler kullanılır.
KeepIdentity	Kaynak tablodaki identity değerleri hedef tablodada korunur. Yani değiştirilmeden eklenir.
KeepNulls	Null değerlerin hedef tabloya korunarak geçirilmesini sağlar.
TableLock	Bulk-Data kopylama işlemi sırasında tabloya kilit koyar.

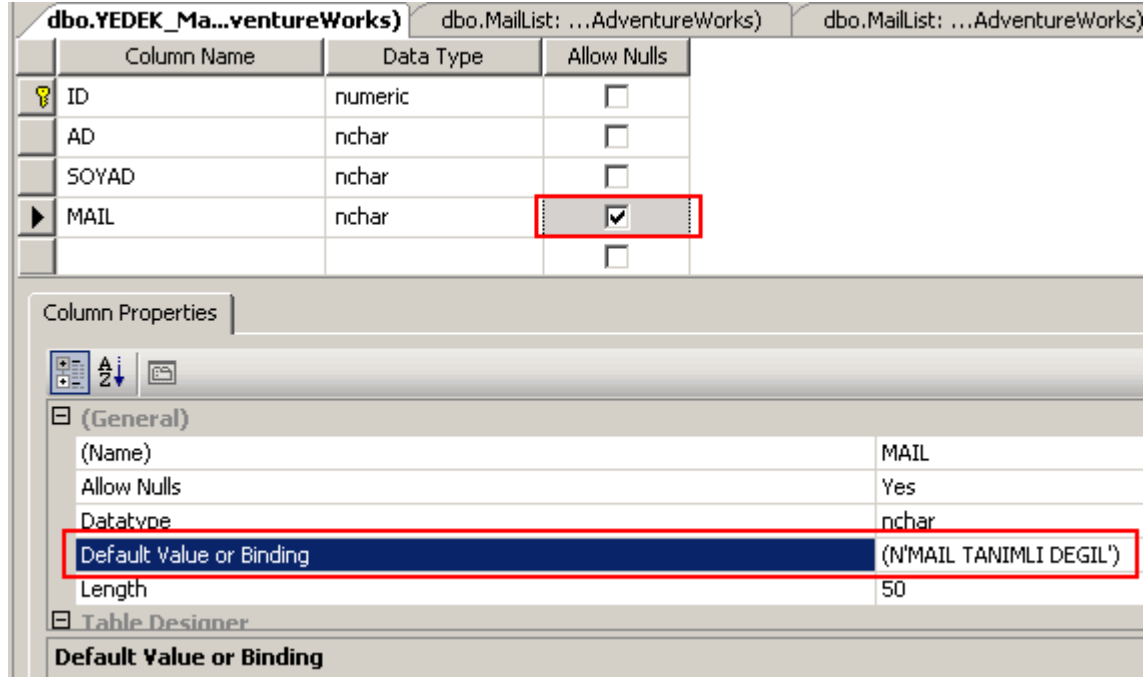
Bizim örneğimizde kullanmamız gereken değer, 'KeepIdentity' dir. Şimdi kodumuzdaki SqlBulkCopy nesnemizin yapıcı metodunu aşağıdaki parametreleri ile çağıralım.

```
SqlBulkCopy bc = new SqlBulkCopy("data source=localhost;initial catalog=AdventureWorks;integrated security=SSPI",  
SqlBulkCopyOptions.KeepIdentity);
```

Uygulamamızı şimdi tekrar çalıştıracak olursak, hedef tablomuz olan YEDEK_MailList içindeki ID alanlarının değerlerinin, kaynak tablodaki ile aynı olduğunu bir başka deyişle korunduğunu görürüz. Uygulamamızı bir kaç sefer üst üste çalıştırsakta sonuç öncekinde olduğu gibi değişmeyecek ve kaynak tablodaki ID alanlarının değerleri hedef tabloya korunarak geçecektir.

Veri taşımalarındaki diğer bir noktada, null değerlerin hedef tabloya taşınmasıdır. Dikkat edecek olursanız, SqlBulkCopyOptions numaralandırıcısı Null değerlerin hedef tabloya korunarak geçirilmesini sağlayan KeepNulls değerine sahiptir. Bu özelliği daha iyi

anlayabilmek amacıyla hedef tablomuzda ufak bir değişiklik yapalım. MAIL alanının null değer içerebildiğini biliyoruz. Hedef tablomuzda bu alan için bir default değer verelim.



Şekil 4. Default Değer belirledik.

Şimdi bu koşullarda MailList isimli kaynak tablomuzda yeni bir satır oluşturalım ancak MAIL alanını null olarak bırakalım. Uygulamayı çalıştırdığımızda, bu satırın null değerinin hedef tabloya taşınmadığını ve hedef tabloya "MAIL TANIMLI DEGIL" değerinin yazıldığını görürüz.

	ID	AD	SOYAD	MAIL
▶	1	Burak Selim	... ŞENYURT	... selim@bsenyurt...
	2	Sefer	... ALGAN	... algans@csharpn...
	3	Okan	... BARLAS	... okanb@csharpn...
	4	Bili	... Amca	... MAIL TANIMLI D...
*				

Şekil 5. Null değer taşınmadı.

Ancak SqlBulkCopy nesnemizi aşağıdaki gibi oluşturduğumuzda, null değerın hedef tablodaki alana taşındığını görürüz.

```
SqlBulkCopy bc = new SqlBulkCopy("data source=localhost;initial catalog=AdventureWorks;integrated security=SSPI",
SqlBulkCopyOptions.KeepNulls);
```

Şimdiye kadar ki kodlarımızda, SqlBulkCopy nesnesi için iki yapıcı metod kullandık. SqlBulkCopy nesnelerini oluşturabileceğimiz yapıcı metodların tamamı aşağıdaki tabloda yer almaktadır.

Name	Description
SqlBulkCopy (SqlConnection)	Hedef bağlantıyı bir SqlConnection nesnesi belirtir.
SqlBulkCopy (String)	Hedef bağlantı için SqlConnection String kullanılır.
SqlBulkCopy (String, SqlBulkCopyOptions)	Hedef bağlantıyı string bilgisi olarak alır ve SqlBulkCopy nesnesini, SqlBulkCopyOptions numaralandırıcısı ile belirtilen şartlara göre oluşturulur.
SqlBulkCopy (SqlConnection, SqlBulkCopyOptions, SqlTransaction)	SqlBulkCopy nesnesini hedef bağlantıda, SqlBulkCopyOptions ile belirtilen şartlarda, SqlTransaction nesnesi ile belirtilen Transaction içinde oluşturur.

Bulk-Data kopyalama işlemini asıl gerçekleştiren WriteToServer metodunda çeşitli aşırı yüklenmiş (overload) versiyonları vardır. Bu versiyonlar aşağıdaki tabloda olduğu gibidir.

Overload Versiyonu	Açıklaması
SqlBulkCopy.WriteToServer (DataRow[])	Bir DataRow dizisini hedef tabloya taşır.
SqlBulkCopy.WriteToServer (DataTable)	Bir DataTable' ı hedef tabloya taşır.
SqlBulkCopy.WriteToServer (IDataReader)	Bir SqlDataReader' dan okuduğu veri kümesini hedef tabloya taşır.
SqlBulkCopy.WriteToServer (DataTable, DataRowState)	Bir DataTable' dan DataRowState numaralandırıcısını belirttiği kriterlere uyan satırlarını, hedef tabloya taşır.

Gördüğümüz gibi, WriteToServer metodunu etkili versiyonları vardır. Örneğin, bir DataTable üzerinde sadece değiştirilmiş olan satırların hedef tabloya yazılmasını sağlayabiliriz. Bunun için tek yapmamız gereken, DataRowState numaralandırıcısının Modified değerini kullanmak olacaktır. Bildiğiniz gibi DataRowState numaralandırıcısı, Modified, Unchanged, Inserted ve Deleted değerlerinden birisini alabilir. Bu durumu derseniz bir örnek üzerinde inceleyelim.

```
static void Main(string[] args)
{
    SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated security=SSPI");
    /*Hedef tablodaki satırları önlem olarak siliyoruz.*/
    SqlCommand cmd = new SqlCommand("DELETE FROM YEDEK_MailList", con);
    con.Open();
    cmd.ExecuteNonQuery();

    SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM MailList", con);
```

```

DataTable dt = new DataTable();
da.Fill(dt);
DataRow drYeni;
drYeni = dt.NewRow();
drYeni["ID"] = "10";
drYeni["AD"] = "Maykıl";
drYeni["SOYAD"] = "Cordin";
drYeni["MAIL"] = "michael@jordan.com";
dt.Rows.Add(drYeni);

```

```

drYeni = dt.NewRow();
drYeni["ID"] = "11";
drYeni["AD"] = "Kerim Abdul";
drYeni["SOYAD"] = "Cabbar";
drYeni["MAIL"] = "nba@nba.com";
dt.Rows.Add(drYeni);

```

```

/*SqlBulkCopy nesnemizi hedef bağlantıyı belirterek oluşturuyoruz.*/

```

```

SqlBulkCopy bc = new SqlBulkCopy("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI",SqlBulkCopyOptions.KeepIdentity);

```

```

/* Hedef tabloyu belirtiyoruz.*/

```

```

bc.DestinationTableName = "YEDEK_MailList";

```

```

/*WriteToServer metodu ile DataTable içinde sadece yeni eklenmiş olan satırları, hedef tabloya yazdırıyoruz.*/

```

```

bc.WriteToServer(dt, DataRowState.Added);

```

```

/* Nesneleri kapatıyoruz.*/

```

```

bc.Close();

```

```

con.Close();

```

```

}

```

Bu kodda gördüğünüz gibi, MailList tablomuzun içeriğini bağlantısız katman nesnelerinden birisi olan DataTable' a aktardık. Daha sonra ise basit bir şekilde iki adet satır ekledik. Bu satırlar yeni eklendikleri için, DataRowState değerleri Added olarak belirlenmiştir. SqlBulkCopy nesnemizin WriteToServer metodunu uygularken,

```

bc.WriteToServer(dt, DataRowState.Added);

```

ikinci parametreye DataRowState.Added değerini verdik. Böylece hedef tabloya, dataTable nesnesinin sahip olduğu veri kümesinden sadece yeni eklenen satırlar yazılacaktır. Uygulamamızı çalıştırdığımızda YEDEK_MailList tablomuzun aşağıdaki satırlara sahip olduğunu görürüz. Bu satırlar uygulamada eklenen satırlardır.

dbo.YEDEK_Ma...ventureWorks)		dbo.YEDEK_Ma...ventureWorks)		dbo.MailList:	
ID	AD	SOYAD	MAIL		
10	Maykıl	... Cordin	... michael@jordan.com ...		
11	Kerim Abdul	... Cabbar	... nba@nba.com ...		
*					

Şekil 5. Sadece DataTable' a eklenen satırların hedef tabloya aktarılması sonucu.

Görüldüğü gibi SqlBulkCopy nesnesi sayesinde, verilerin başka tablolara aktarılması son derece kolaydır. Bununla birlikte, bu tekniği yedekleme işlemleri için sıklıkla kullanabiliriz. Herşeyden önce, var olan bir tablonun tüm satırlarını, satırların belirli bir kümesini, yada son kullanıcı tarafından bağlantısız katmanda yeni eklenmiş, silinmiş, güncellenmiş satırların tamamını başka bir hedef tabloya kolayca aktarabilme yeteneğine sahip olmuş oluyoruz.

Şu ana kadar gerçekleştirdiğimiz örnekler, aynı SqlConnection' ı kullanmaktadır. Oysaki gerçek hayatta bu tip taşıma işlemlerini, farklı sunucularda yer alan farklı tablolara doğru gerçekleştirmek isteyebiliriz. Bununla birlikte, hedef tablomuz nerede olursa olsun farklı bir şema yapısınada sahip olabilir. Böyle bir durumda özellikle alanların doğru ve uygun bir şekilde eşleştirilmesi önem kazanmaktadır. Şimdi bu durumu incelemek amacıyla öncelikle, YEDEK_MailList tablomuzu farklı alan isimleri ve sıralama ile aynı sunucuda bulunan farklı bir veritabanı altına koyalım.

Column Name	Data Type	Allow Nulls
UserID	numeric	☐
MailAddress	nchar	☒
UserName	nchar	☐
UserLastName	nchar	☐
AccountStatus	nchar	☒

Şekil 7. YEDEK_MailList Tablomuz.

Bu sefer tablomuzu hem farklı bir veritabanı altına koyduk, hemde şema yapısını biraz daha değiştirdik. Herşeyden önemlisi, alan adlarını, sıralarını değiştirdik. Bununla birlikte fazladan bir alanımız(AccountStatus) daha mevcuttur. Bu nedenle, kaynak ve hedef tablolardaki alanları birbirleriyle eşleştirmemiz gerekmektedir. İşte bu amaçla, kodlarımızda SqlBulkCopyColumnMapping sınıfına ait nesne örneklerini kullanmalıyız. Şimdi uygulama kodlarımızı aşağıdaki gibi değiştirelim.

```
static void Main(string[] args)
{
    try
```

```

{
    SqlConnection con = new SqlConnection("data source=localhost;initial catalog=Dukkan;integrated security=SSPI");
    /*Hedef tablodaki satırları önlem olarak siliyoruz.*/
    SqlCommand cmd = new SqlCommand("DELETE FROM YEDEK_MailList", con);
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();

    /* Kaynak tablodan verileri SqlDataReader yardımıyla çekiyoruz.*/
    SqlConnection con1 = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI");
    SqlCommand cmdCek = new SqlCommand("SELECT * FROM MailList",con1);
    SqlDataReader dr;
    con1.Open();
    dr = cmdCek.ExecuteReader();

    /*SqlBulkCopy nesnemizi hedef bağlantıyı belirterek oluşturuyoruz.*/
    SqlBulkCopy bc = new SqlBulkCopy("data source=localhost;initial catalog=Dukkan;integrated security=SSPI",
SqlBulkCopyOptions.KeepIdentity);
    /* Hedef tabloyu belirtiyoruz.*/
    bc.DestinationTableName = "YEDEK_MailList";

    /* Kaynak ve Hedef tabloların alanları eşleştiriliyor.*/
    SqlBulkCopyColumnMapping cm1 = new SqlBulkCopyColumnMapping("ID", "UserID");
    SqlBulkCopyColumnMapping cm2 = new SqlBulkCopyColumnMapping("AD", "UserName");
    SqlBulkCopyColumnMapping cm3 = new SqlBulkCopyColumnMapping("SOYAD", "UserLastName");
    SqlBulkCopyColumnMapping cm4 = new SqlBulkCopyColumnMapping("MAIL", "MailAddress");
    /* Eşleştirmeler için oluşturulan SqlBulkCopyColumnMapping nesneleri, SqlBulkCopy nesnemizin ColumnMappings
koleksiyonuna ekleniyor.*/
    bc.ColumnMappings.Add(cm1);
    bc.ColumnMappings.Add(cm2);
    bc.ColumnMappings.Add(cm3);
    bc.ColumnMappings.Add(cm4);

    /*WriteToServer metodu ile SqlDataReader' dan okuduğumuz satırları, hedefe insert ediyoruz.*/
    bc.WriteToServer(dr);

    /* Nesneleri kapatıyoruz.*/
    bc.Close();

```

```

        dr.Close();
        con1.Close();
    }
    catch (SqlException ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
}

```

Burada bizim için önemli olan, kaynak ve hedef tablolardaki alan eşleştirmesini nasıl bildirdiğimizdir. Bunun için, aşağıdaki örnek söz dizimini kullandık.

```
SqlBulkCopyColumnMapping cm1 = new SqlBulkCopyColumnMapping("ID", "UserID");
```

Örneğimizde, SqlBulkCopyColumnMapping sınıfının aşağıdaki prototipe sahip olan yapıcısını kullandık.

```
public SqlBulkCopyColumnMapping(string sourceColumn, string destinationColumn);
```

Bu yapıcı haricinde kullanabileceğimiz diğer yapıcılarda aşağıdaki tabloda yer almaktadır.

Yapıcı Metod	Açıklama
SqlBulkCopyColumnMapping ()	
SqlBulkCopyColumnMapping (Int32, Int32)	Kaynak ve Hedef alanların indeksini alır.
SqlBulkCopyColumnMapping (Int32, String)	Kaynak alanın indeksini, hedef alanın ismini alır.
SqlBulkCopyColumnMapping (String, Int32)	Kaynak alanın ismini, hedef alanın indeksini alır.

SqlBulkCopyColumnMapping nesnelerimizi yarattıktan sonra bu nesnelerin, SqlBulkCopy nesnemizin ilgili ColumnMappings koleksiyonuna eklenmesi gerekmektedir. Bunun için Add metodunu kullandık. Böylece SqlBulkCopy nesnemiz, kaynak tablodaki hangi alanın, hedef tablodaki hangi alana denk geleceğini bilmektedir. Aynı örneği, aşağıdaki kodlar ilede gerçekleştirebiliriz.

```

bc.ColumnMappings.Add("ID", "UserID");
bc.ColumnMappings.Add("AD", "UserName");
bc.ColumnMappings.Add("SOYAD", "UserLastName");
bc.ColumnMappings.Add("MAIL", "MailAddress");

```

Burada daha kısa bir çözüm uygulanmıştır. Bu sefer doğrudan ColumnMappings özelliğinin Add metodu kullanılarak, ilgili alan eşleştirmeleri SqlBulkCopy nesnesine eklenmiştir. Bir SqlBulkCopyColumnMapping nesnesinin oluşturulmasında kullanılan aşırı yüklenmiş yapıcılardaki parametreler buradaki Add metodu içinde geçerlidir. Yani dilersek, Add metodunda alanların indeks değerlerini de kullanabiliriz. Örneğimizi çalıştırdığımızda, kaynak tablodaki satırların yeni tablomuza başarılı bir şekilde eklendiğini görürüz.

dbo.YEDEK_Mai...(burki.Dukkan)		dbo.YEDEK_Mai...(burki.Dukkan)		Program.cs	
	UserID	MailAddress	UserName	UserLastName	AccountStatus
▶	1	selim@bsenyurt...	Burak Selim	... ŞENYURT	... NULL
	2	algans@csharpn...	Sefer	... ALGAN	... NULL
	3	okanb@csharpn...	Okan	... BARLAS	... NULL
	4	NULL	Bili	... Amca	... NULL
*					

Şekil 8. Alanların eşleştirilmesi sonucu.

SqlBulkCopy sınıfının SqlRowsCopied isimli tek bir olayı vardır. Bu olay, SqlBulkCopy sınıfının NotifyAfter özelliğine verilen değeri baz alır ve bu değeri kullanarak Bulk-Data kopyalama işlemleri sırasında periyodik olarak çalışır. Yani, eğer biz NotifyAfter özelliğine 20 değerini atarsak, kaynaktan hedefe taşınan her 20 satırda bir SqlRowsCopied olayı çalışacaktır. Bu olayın prototipi aşağıdaki gibidir.

```
public event SqlRowsCopiedEventHandler SqlRowsCopied;
```

Olayın gerçekleşmesi sonucu çalıştırılacak metodu, SqlRowsCopiedEventHandler delegatesi aşağıdaki gibi tanımlar.

```
public sealed delegate void SqlRowsCopiedEventHandler(object sender, SqlRowsCopiedEventArgs e);
```

Şimdi örneğimizdeki SqlBulkCopy nesnesi için SqlRowsCopied olayını kullanalım.

```
static void Main(string[] args)
{
    try
    {
        .
        .
        .
        SqlBulkCopy bc = new SqlBulkCopy("data source=localhost;initial catalog=Dukkan;integrated security=SSPI",
        SqlBulkCopyOptions.KeepIdentity);
        bc.DestinationTableName = "YEDEK_MailList";
        bc.ColumnMappings.Add("ID", "UserID");
        bc.ColumnMappings.Add("AD", "UserName");
        bc.ColumnMappings.Add("SOYAD", "UserLastName");
        bc.ColumnMappings.Add("MAIL", "MailAddress");

        bc.NotifyAfter = 1;
        bc.SqlRowsCopied += new SqlRowsCopiedEventHandler(bc_SqlRowsCopied);
        bc.WriteToServer(dr);
        .
        .
    }
}
```



```

    }
    catch (SqlException ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
}

```

```

static void bc_SqlRowsCopied(object sender, SqlRowsCopiedEventArgs e)

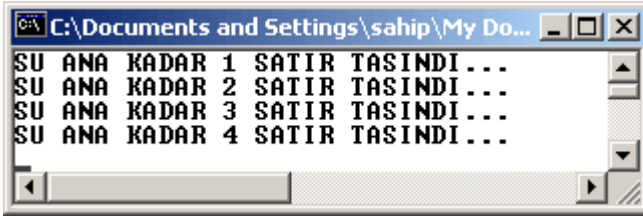
```

```

{
    Console.WriteLine("ŞU ANA KADAR "+e.RowsCopied.ToString() + " SATIR TAŞINDI...");
}

```

Burada, SqlBulkCopy nesnemize SqlRowsCopied olayını ekledikten sonra bu olayın gerçekleşmesi sonucu çalıştırılacak olan bc_SqlRowsCopied metodunda, o ana kadar taşınmış olan satır sayısını ekrana yazdırıyoruz. NotifyAfter özelliğine 1 değerini atadığımız için, olay metodumuz her bir satır hedef tabloya başarılı bir şekilde taşındıktan sonra tetiklenecektir. Sonuç aşağıdaki gibi olur.



Şekil 9. SqlRowsCopied olayının işlenmesi sonucu.

Bu makalemizde, Ado.Net 2.0 ile gelen yeni kabiliyetlerden birisine değinmeye çalıştık. İlerleyen makalelerimizde Ado.Net 2.0' ın yeni özelliklerini incelemeye devam edeceğiz. Hepinize mutlu günler dilerim.

Ado.Net 2.0 ve Toplu Güncelleme İşlemleri (Batch-Updates)

Değerli Okurlarım Merhabalar,

Toplu güncelleştirme işlemleri, birden fazla sql ifadesinin (insert,update,delete,select gibi) arka arkaya gelecek şekilde ancak tek bir seferde çalıştırılmasını baz alan bir tekniktir. Ado.Net 2.0 ile, toplu güncelleştirme işlemlerine daha fazla fonksiyonellik kazandırılmıştır. Bu koşul elbetteki toplu güncelleştirme işlemlerini destekleyen veritabanı sunucuları üzerinde geçerli olmaktadır. Şu an için, yönetimsel kodda yer alan Oracle ve Sql nesnelerinin desteklediği bu fonksiyonelliği kazanmak için aşağıda prototipi verilen ve SqlDataAdapter yada OracleDataAdapter sınıflarına ait olan, UpdateBatchSize özelliği kullanılmaktadır.

```

public override int UpdateBatchSize {get;set;}

```

Bu özellik bir anlamda, DataAdapter nesnesinin Update komutu ile veritabanına doğru yapılacak güncelleme işlemlerinin toplu olarak hangi periyotta gerçekleştirileceğini belirtir. Örneğin, 1 varsayılan değeridir ve her bir güncelleme işleminin (insert,update

veya delete) her satır için ayrı ayrı yapılacağını belirtir. Daha derin düşünecek olursa, örneğin Sql Sunucusunda yer alan sp_executesql stored procedure' ünün her bir satır için birer kez ilgili komutu (Insert gibi) çalıştıracağını belirtir.

Diğer yandan, bu özelliğe 0 değerini verdiğimizde tüm güncelleme işlemleri tek bir seferde gerçekleştirilir. Bir başka deyişle veritabanına doğru n sayıda güncelleme işlemi varsa, Sql Sunucusunda yer alan sp_executesql stored procedure' ü bu n sayıdaki işlemleri içeren toplu bir komut kümesini tek bir seferde çalıştıracaktır. Ayrıca UpdateBatchSize özelliğine 0 ve 1 haricinde verilecek olan pozitif değerler, her bir toplu güncelleştirme işleminin kaç iç komut içereceğini belirtmektedir. Konuyu daha kolay bir şekilde anlayabilmek için basit bir Console uygulaması geliştirelim.

```
#region Using directives
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;
```

```
#endregion
```

```
namespace BatchUpdates
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI");
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM MailList", con);
            DataTable dt = new DataTable();
            da.Fill(dt);

            DataRow dr;
            for (int i = 1; i <= 5; i++)
            {
                dr = dt.NewRow();
                dr["AD"] = "AD_" + i.ToString();
                dr["SOYAD"] = "SOYAD_" + i.ToString();
                dr["MAIL"] = "MAIL_" + i.ToString();
                dt.Rows.Add(dr);
            }
        }
    }
}
```

```

da.UpdateBatchSize = 1;
SqlCommandBuilder cm = new SqlCommandBuilder(da);
da.Update(dt);
Console.WriteLine("İŞLEMLERİN SONU");
Console.ReadLine();
}
}
}

```

Bu uygulamada ADO.NET 1.1 ile yapabildiğimiz işlemlerden farklı bir şey yoktur. Yukarıda yer alan MailList tablomuzda SqlDataAdapter nesnesi vasıtasıyla 5 adet satır giriyoruz. Bizim için önemli olan UpdateBatchSize değerinin 1 olarak belirtilmesidir. Uygulamamızı çalıştırmadan önce, SQL Profiler aracını kullanarak yeni bir Trace başlatalım ve SQL Sunucumuzda gerçekleşen işlemleri izlemeye çalışalım. Trace'imiz çalışırken uygulamamızı yürütecek olursak, SQL Sunucusu üzerinde aşağıdaki olayların gerçekleştirildiğini görürüz.

EventClass	SP...	TextData	NTUser...	ClientProcess...	ApplicationN...
SQL:BatchCompleted	56	SELECT * FROM MailList	sahip	2372	.Net SqlClier
RPC:Completed	56	exec sp_resetconnection	sahip	2372	.Net SqlClier
SQL:BatchCompleted	56	SET FMTONLY OFF; SET NO_BR...	sahip	2372	.Net SqlClier
RPC:Completed	56	exec sp_executesql N'INSERT INTO...	sahip	2372	.Net SqlClier
RPC:Completed	56	exec sp_executesql N'INSERT INTO...	sahip	2372	.Net SqlClier
RPC:Completed	56	exec sp_executesql N'INSERT INTO...	sahip	2372	.Net SqlClier
RPC:Completed	56	exec sp_executesql N'INSERT INTO...	sahip	2372	.Net SqlClier
RPC:Completed	56	exec sp_executesql N'INSERT INTO...	sahip	2372	.Net SqlClier
Audit Logout	56		sahip	2372	.Net SqlClier
SQL:BatchCompleted	53	SELECT N'Testing Connection...'	SYSTEM	1224	SQLAgent -
SQL:BatchCompleted	53	EXECUTE msdb.dbo.sp_sqlagent_g...	SYSTEM	1224	SQLAgent -
SQL:BatchCompleted	53	SELECT N'Testing Connection...'	SYSTEM	1224	SQLAgent -
SQL:BatchCompleted	53	EXECUTE msdb.dbo.sp_sqlagent_g...	SYSTEM	1224	SQLAgent -

```

exec sp_executesql N'INSERT INTO [MailList] ([AD], [SOYAD], [MAIL]) VALUES (@p1, @p2, @p3)' N'@p1 nchar(4),@p2 nchar(7)
go
exec sp_executesql N'INSERT INTO [MailList] ([AD], [SOYAD], [MAIL]) VALUES (@p1, @p2, @p3)' N'@p1 nchar(4),@p2 nchar(7)
go
exec sp_executesql N'INSERT INTO [MailList] ([AD], [SOYAD], [MAIL]) VALUES (@p1, @p2, @p3)' N'@p1 nchar(4),@p2 nchar(7)
go
exec sp_executesql N'INSERT INTO [MailList] ([AD], [SOYAD], [MAIL]) VALUES (@p1, @p2, @p3)' N'@p1 nchar(4),@p2 nchar(7)
go
exec sp_executesql N'INSERT INTO [MailList] ([AD], [SOYAD], [MAIL]) VALUES (@p1, @p2, @p3)' N'@p1 nchar(4),@p2 nchar(7)
go

```

Trace is running. Ln 15, Col 1 Rows: 24

Şekil 1. UpdateBatchSize değeri 1 olduğunda.

Dikkat edecek olursanız, `sp_executesql` stored procedure' ü girilen her satır için insert komutunu birer kez çalıştırmıştır. Bunun nedeni `UpdateBatchSize` özelliğinin 1 değerine sahip olmasıdır. Eğer bu değeri 0 yapıp tekrar çalıştırsak, bu takdirde kaç satır girersek girerlim tüm satırlar için geçerli olan insert komutları tek bir toplu-komut olarak işlenecek ve tek bir seferde çalıştırılacaktır. Örnek olarak, döngümüzün değerini 20 satır insert edilecek şekilde ayarladığımızı düşünersek, `UpdateBatchSize` özelliğine 0 değerini vermek ile, 20 satır için parametre alacak tek bir stored procedure' ü çağırmış oluruz. Uygulamamızda bu kez tüm satırları update ettiğimizi düşünelim ve `UpdateBatchSize` özelliğine 0 değerini verelim.

```
for (int i = 0; i < dt.Rows.Count; i++)  
{  
    dr = dt.Rows[i];  
    dr["AD"] = "_DEGISTI";  
}
```

da.UpdateBatchSize = 0;

Şimdi Sql Profiler' da Trace' imizdeki işlemlere bakacak olursa, kaç satır güncellenmiş ise her bir satır için yapılan update işlemlerinin tek bir toplu-komut kümesinde gerçekleştirildiğini görürüz.

Trace_Batch_Updates (BURKI)			
EventClass	SP...	TextData	NTUse
RPC:Completed	57	exec sp_resetconnection	sahip
SQL:BatchCompleted	57	SET FMTONLY OFF; SET NO_BR...	sahip
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip
Audit Logout	57		sahip
SQL:BatchCompleted	53	SELECT N'Testing Connection...'	SYSTE
SQL:BatchCompleted	53	EXECUTE msdb.dbo.sp_sqlagent_g...	SYSTE


```

exec sp_executesql N'BEGIN
exec sp_executesql @P1, @P2, @P3, @P4, @P5, @P6, @P7, @P8;
exec sp_executesql @P1, @P2, @P9, @P10, @P11, @P12, @P13, @P14;
exec sp_executesql @P1, @P2, @P15, @P16, @P17, @P18, @P19, @P20;
exec sp_executesql @P1, @P21, @P22, @P23, @P24, @P25, @P26, @P27;
exec sp_executesql @P1, @P2, @P28, @P29, @P30, @P31, @P32, @P33;
exec sp_executesql @P1, @P2, @P34, @P35, @P36, @P37, @P38, @P39;
exec sp_executesql @P1, @P2, @P40, @P41, @P42, @P43, @P44, @P45;
exec sp_executesql @P1, @P2, @P46, @P47, @P48, @P49, @P50, @P51;
exec sp_executesql @P1, @P2, @P52, @P53, @P54, @P55, @P56, @P57;
exec sp_executesql @P1, @P2, @P58, @P59, @P60, @P61, @P62, @P63;
exec sp_executesql @P1, @P2, @P64, @P65, @P66, @P67, @P68, @P69;
exec sp_executesql @P1, @P2, @P70, @P71, @P72, @P73, @P74, @P75;
exec sp_executesql @P1, @P2, @P76, @P77, @P78, @P79, @P80, @P81;
exec sp_executesql @P1, @P2, @P82, @P83, @P84, @P85, @P86, @P87;
exec sp_executesql @P1, @P2, @P88, @P89, @P90, @P91, @P92, @P93;
exec sp_executesql @P1, @P2, @P94, @P95, @P96, @P97, @P98, @P99;
exec sp_executesql @P1, @P2, @P100, @P101, @P102, @P103, @P104, @P105;
exec sp_executesql @P1, @P2, @P106, @P107, @P108, @P109, @P110, @P111;
exec sp_executesql @P1, @P2, @P112, @P113, @P114, @P115, @P116, @P117;
exec sp_executesql @P1, @P2, @P118, @P119, @P120, @P121, @P122, @P123;
exec sp_executesql @P1, @P2, @P124, @P125, @P126, @P127, @P128, @P129;
exec sp_executesql @P1, @P2, @P130, @P131, @P132, @P133, @P134, @P135;
exec sp_executesql @P1, @P2, @P136, @P137, @P138, @P139, @P140, @P141;
exec sp_executesql @P1, @P2, @P142, @P143, @P144, @P145, @P146, @P147;

```

Trace is running.

Şekil 2. UpdateBatchSize özelliğine 0 değeri verildiğinde.

Elbette daha önceden bahsettiğimiz gibi UpdateBatchSize özelliğine 0 ve 1 haricinde pozitif değerlerde verebiliriz. Bu durumda toplu-komut kümeleri belirtilen sayı kadar iç komut içerecektir. Örneğimizde, UpdateBatchSize değerini 7 yaparsak, her bir sp_executesql çağrısında, içsel olarak 7 satırlık işlem içeren toplu-komut kümeleri olduğunu görürüz.

da.**UpdateBatchSize = 7;**

Trace_Batch_Updates (BURKI)						
EventClass	SP...	TextData	NTUser...	ClientProcess...	Application	
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip	2252	.Net SqlCli	
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip	2252	.Net SqlCli	
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip	2252	.Net SqlCli	
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip	2252	.Net SqlCli	
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip	2252	.Net SqlCli	
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip	2252	.Net SqlCli	
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip	2252	.Net SqlCli	
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip	2252	.Net SqlCli	
RPC:Completed	57	exec sp_executesql N'BEGIN exec s...	sahip	2252	.Net SqlCli	
Audit Logout	57		sahip	2252	.Net SqlCli	
SQL:BatchCompleted	53	SELECT N'Testing Connection...'	SYSTEM	1224	SQLAgent	
SQL:BatchCompleted	53	EXECUTE msdb.dbo.sp_sqlagent_g...	SYSTEM	1224	SQLAgent	
SQL:BatchCompleted	53	SELECT N'Testing Connection...'	SYSTEM	1224	SQLAgent	
SQL:BatchCompleted	53	EXECUTE msdb.dbo.sp_sqlagent_g...	SYSTEM	1224	SQLAgent	
SQL:BatchCompleted	53	SELECT N'Testing Connection...'	SYSTEM	1224	SQLAgent	
SQL:BatchCompleted	53	EXECUTE msdb.dbo.sp_sqlagent_g...	SYSTEM	1224	SQLAgent	
SQL:BatchCompleted	53	SELECT N'Testing Connection...'	SYSTEM	1224	SQLAgent	


```

exec sp_executesql N'BEGIN
exec sp_executesql @P1, @P2, @P3, @P4, @P5, @P6, @P7, @P8;
exec sp_executesql @P1, @P2, @P9, @P10, @P11, @P12, @P13, @P14;
exec sp_executesql @P1, @P2, @P15, @P16, @P17, @P18, @P19, @P20;
exec sp_executesql @P1, @P2, @P21, @P22, @P23, @P24, @P25, @P26;
exec sp_executesql @P1, @P2, @P27, @P28, @P29, @P30, @P31, @P32;
exec sp_executesql @P1, @P2, @P33, @P34, @P35, @P36, @P37, @P38;
exec sp_executesql @P1, @P2, @P39, @P40, @P41, @P42, @P43, @P44;
END
'N'@P1 ntext,@P2 ntext,@P3 nchar(8),@P4 decimal(18,0),@P5 nchar(50),@P6 nchar(50),@P7 int,@P8 nchar(50),@P9 nchar(

```

Şekil 3. UpdateBatchSize değerini pozitif her hangibir sayı olarak belirlediğimizde.

Örneklerdende görüldüğü gibi, ADO.NET 2.0 toplu-komut güncelleme işlemlerine daha fazla fonksiyonellik katmak amacıyla kullanışlı bir özellik kazanmıştır. Bazı durumlarda, güncelleme işlemlerinin bağlantısız katmandan, veritabanına doğru olan hareketlerinde toplu olarak yapılması network trafiğini olumlu yönde etkileyecek bir gelişmedir. Çünkü, tüm güncelleme hareketleri için veritabanına doğru sadece tek bir tur atılacaktır. Elbetteki devasa boyutlara sahip olan veri kaynakları üzerinde yapılacak büyük çaplı güncelleme işlemlerinde, toplu-komut kümelerini belirli sayılarda komut içerecek şekilde ayarlamakta performans açısından olumlu bir etki yaratacaktır.

Bu makalemizde, kısaca toplu-güncelleştirme (Batch-Update) işlemlerine değinmeye çalıştık. İlerleyen makalelerimizde, ADO.NET 2.0' in yeni özelliklerine bakmaya devam edeceğiz. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

ADO.NET 2.0 ile Mars' a Ayak Basıyoruz

Değerli Okurlarım Merhabalar,

Bu makalemizde, MARS (Multiple Active Results Sets) kavramını incelemeye çalışacağız. MARS kavramı Ado.Net 2.0 mimarisine monte edilmiş yeni bir yapıdır. Bu yapının bize sağladığı avantajları anlayabilmek için, Ado.Net 1.0/1.1 sürümlerinin kabiliyetlerine ve kısıtlamalarına kısaca bir göz atmak gerekmektedir.

Ado.Net' in önceki sürümlerinde, özellikle veri kümelerini uygulama ortamına çekmek istediğimizde kullanabileceğimiz iki temel teknik vardır. Birincisi, bağlantısız katmana veri alabilmek için DataAdapter, DataTable, DataSet gibi nesnelerin kullanılmasıdır. İkincisi ise, daha süratli veri çekmemizi sağlayan ve veri kaynağına olan bağlantının sql komutunun işleyişi boyunca açık olmasını gerektiren, DataReader nesnelerinin kullanıldığı tekniktir.

Bu tekniklerin hangisi kullanılırsa kullanılsın özellikle aynı bağlantı üzerinden eş zamanlı olarak yürütülen sql komutlarının çalıştırılmasına izin veren bir yapı yoktur. Yani, veri kaynağından bir DataReader nesnesi ile veri çekerken, aynı açık bağlantı üzerinden güncelleme, ekleme, silme gibi işlemleri içeren sql komutlarını çalıştıramayız. İşte bu imkansızlığı gidermek amacıyla Ado.Net mimarisi içine MARS, Asenkron Komut Yürütme ve ObjectSpace gibi yeni ve köklü değişiklikler eklenmiştir. Biz bu günkü makalemizde, aynı açık bağlantı üzerinden birden fazla kayıt kümesini uygulama ortamına çekmemize ve onlara erişmemize imkan sağlayan MARS (Multiple Active Results Sets) yapısını tanımaya çalışacağız.

MARS, özellikle aynı açık bağlantı üzerinden birden fazla satır kümesini elde edebileceğimiz sql komutlarının eş zamanlı olarak çalıştırılmasına imkan verir. Örneğin Ado.Net' in eski sürümlerinde şu senaryoyu göz önüne alalım; aynı veri kaynağında yer alan ve aynı bağlantı üzerinden erişeceğimiz üç kayıt kümesi olsun. Bu kayıt kümelerini 3 farklı SqlDataReader nesnesi ile ortama çekmek istediğimizi düşünelim. Bu işlemi simüle edebilmek için VS.NET 2003' de basit bir Console uygulaması geliştireceğiz. Uygulamamızın kodları aşağıdaki gibi olacaktır.

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

namespace MultipleReader
{
    class AdoNet1nokta1de
    {
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                SqlConnection con=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=SSPI");

                SqlDataReader dr1;
                SqlDataReader dr2;
                SqlDataReader dr3;
```

```

SqlCommand cmd1=new SqlCommand("SELECT * FROM Customers",con);
SqlCommand cmd2=new SqlCommand("SELECT * FROM Orders",con);
SqlCommand cmd3=new SqlCommand("SELECT * FROM [Order Details]",con);

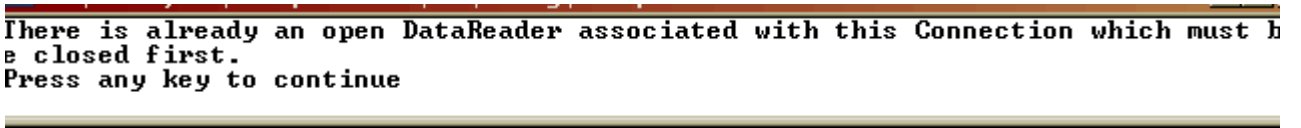
con.Open();

dr1=cmd1.ExecuteReader();
dr2=cmd2.ExecuteReader();
dr3=cmd3.ExecuteReader();

con.Close();
}
catch(Exception hata)
{
    Console.WriteLine(hata.Message);
}
}
}
}

```

Uygulamamızı çalıştırdığımızda çalışma zamanında bir istisnanın fırlatıldığını görürüz.



```

There is already an open DataReader associated with this Connection which must be closed first.
Press any key to continue

```

Şekil 1. ADO.NET 1.0./1.1' deki durum.

Sorun şudur ki, ilk çalıştırılan SqlDataReader nesnesini kapatmadan diğerlerini çalıştırmamız da mümkün değildir. Nitekim SqlDataReader nesnesi, çalıştırdığı SQL komutunu ile verileri çekebilmek için ilgili bağlantının kendisi için tahsis edilmesini ve işi bitene kadar da başkaları tarafından kullanılmamasını gerektirir. Bu, tamamıyla ADO.NET' in mimarisinden kaynaklanan bir güçlüktür. Bununla birlikte, akla şöyle bir çözüm yolu gelebilir. Her bir SqlDataReader nesnesini kendi SqlConnection havuzu içinde çalıştırmak. Yani yukarıdaki kodu aşağıdaki gibi yazmak.

```

SqlConnection con1=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=SSPI");
SqlConnection con2=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=SSPI");
SqlConnection con3=new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=SSPI");
SqlDataReader dr1;
SqlDataReader dr2;
SqlDataReader dr3;

```



```
SqlCommand cmd1=new SqlCommand("SELECT * FROM Customers",con1);
SqlCommand cmd2=new SqlCommand("SELECT * FROM Orders",con2);
SqlCommand cmd3=new SqlCommand("SELECT * FROM [Order Details]",con3);
```

```
con1.Open();
con2.Open();
con3.Open();
```

```
dr1=cmd1.ExecuteReader();
dr2=cmd2.ExecuteReader();
dr3=cmd3.ExecuteReader();
```

```
con1.Close();
con2.Close();
con3.Close();
```

Bu haliyle uygulama çalışacak ve herhangi bir hata mesajı yada istisna vermeyecektir. Uygulanan bu teknik her ne kadar çözümmüş gibi görünsede, gereksiz yere kaynak tüketimine neden olmaktadır. Nitekim aynı bağlantı için üç kez SqlConnection nesnesi örneklendirilmiş ve sql sunucuna üç adet ayrı isimli ama aynı özellikte bağlantı hattı tesis edilmiştir. Dahası bu teknik kullanıldığı takdirde her çalışan Sql komutunun işleyişi sonlanana kadar, bir sonraki sql komutuna geçilemeyecektir. Bunu çözmek içinde, bu veri çekme işlemlerini ayrı metodlar halinde tanımlayıp çok-kanallı (multi-threading) programlama tekniklerini uygulayabiliriz. Ancak bu yaklaşımlar elbetteki çok verimli yada ölçeklenebilir değildir.

Çözüm, .Net mimarları tarafından Ado.Net 2.0' a yerleştirilmiştir ve MARS olarak adlandırılmıştır. Şimdi ilk yazdığımız uygulamayı birde .Net framework 2.0 ortamında geliştirelim. Kodlarımızı hiç değiştirmeyeceğiz. Bu kez Yukon üzerinde yer alan AdventureWorks isimli veritabanındaki iki tabloya aynı açık bağlantı üzerinden aynı zamanda erişmeye çalışacağız. İşte kodlarımız,

```
#region Using directives
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Data;
```

```
#endregion
```

```
namespace MarsEtkisi
{
    class Program
```

```
{
    static void Main(string[] args)
    {
        try
        {
            SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI");

            SqlCommand cmd1 = new SqlCommand("SELECT TOP 10 * FROM Person.address", con);
            SqlCommand cmd2 = new SqlCommand("SELECT TOP 3 * FROM Person.contact", con);

            SqlDataReader dr1;
            SqlDataReader dr2;

            con.Open();

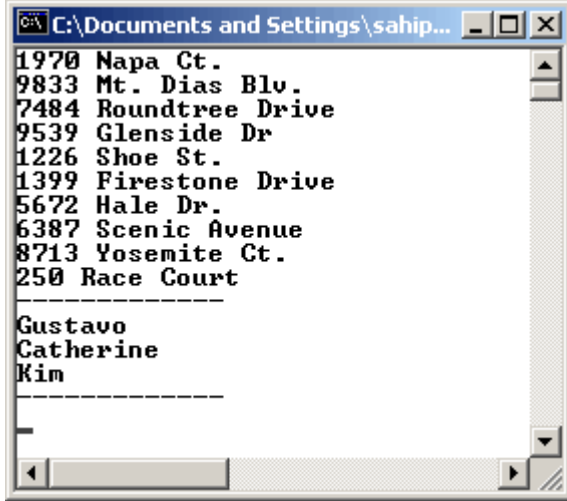
            dr1 = cmd1.ExecuteReader();
            dr2 = cmd2.ExecuteReader();

            while (dr1.Read())
            {
                Console.WriteLine(dr1["AddressLine1"]);
            }
            Console.WriteLine("-----");
            while (dr2.Read())
            {
                Console.WriteLine(dr2["FirstName"]);
            }

            con.Close();
            Console.WriteLine("-----");
            Console.ReadLine();
        }
        catch (SqlException hata)
        {
            Console.WriteLine(hata.Message);
        }
    }
}
```

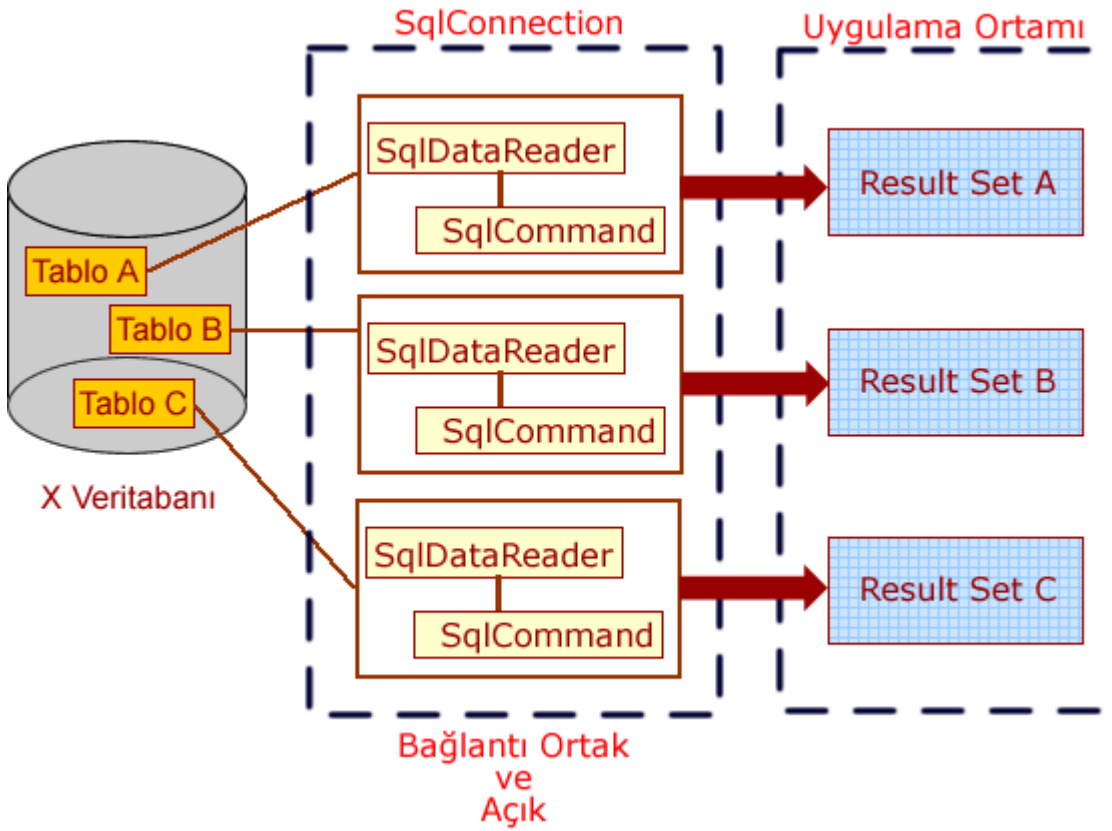
}

Uygulamayı bu haliyle çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz. Görüldüğü gibi, SqlDataReader nesneleri, diğerinin kapatılmasına gerek duymadan veri kaynağından veri çekebilmiştir.



Şekil 2. MARS Etkisi.

Bu örnek basit olarak MARS kavramını açıklamaktadır. Olayı daha iyi kavrayabilmek amacıyla aşağıdaki şekildende faydalanabiliriz.



Şekil 3. MARS (Multiple Active Results Sets)

Şekildende görülebileceği gibi, bir veritabanında yer alan çeşitli sayıdaki tabloya ait veri kümelerini aynı uygulama ortamına, farklı ama tek bir ortak bağlantıyı kullanan DataReader nesneleri yardımıyla erişebilmemiz mümkündür.

Şimdi derseniz, MARS etkisini daha iyi görebileceğimiz başka bir örneği göz önüne alalım. Bu kez bir birleriyle bire-çok ilişkisi olan iki tabloyu ele alacağız. Bu tablolara ait verileri uygulama ortamına yine DataReader nesneleri yardımıyla çekeceğiz. Ancak işin ilginç yanı, bir DataReader nesnesi çalışırken ve satırları arasında iterasyon yaparken, diğerine parametre göndererek her bir satır için diğer DataReader' ında çalıştırılmasının sağlanmış olacaktır. Bu örneği gerçekleştirmek için aşağıdaki kodları yazarak, Visual Studio.Net 2005' de basit bir Console uygulaması geliştirelim.

```
static void Main(string[] args)
{
    try
    {
        SqlConnection con = new SqlConnection("data source=localhost;initial catalog=Dukkan;integrated security=SSPI");

        SqlCommand cmd1 = new SqlCommand("SELECT * FROM DepartmanSinif", con);
        SqlCommand cmd2 = new SqlCommand("SELECT * FROM PersonelDetay WHERE PersonelID=@PersonelID", con);
        cmd2.Parameters.Add("@PersonelID", SqlDbType.Int);

        SqlDataReader dr1;
        SqlDataReader dr2;

        con.Open();
        dr1 = cmd1.ExecuteReader();
        while (dr1.Read())
        {
            Console.WriteLine(dr1["Departman"]);
            Console.WriteLine(" ALTINDAKI ELEMANLARI....");
            cmd2.Parameters["@PersonelID"].Value = dr1["PersonelID"];
            dr2 = cmd2.ExecuteReader();
            while (dr2.Read())
            {
                Console.WriteLine(dr2["Ad"]+" " +dr2["Soyad"]+" " +dr2["Mail"]);
            }
            dr2.Close();
            Console.WriteLine("-----");
        }

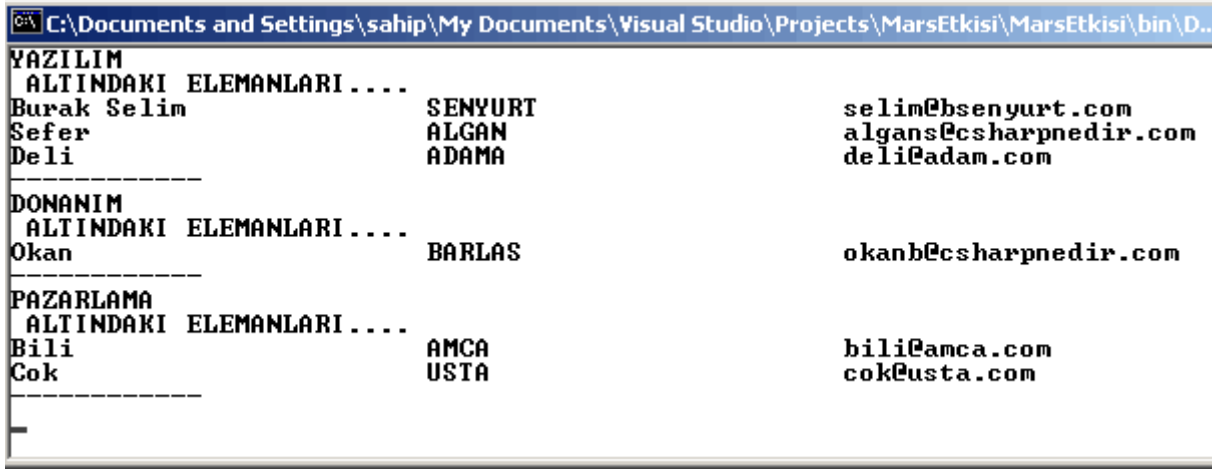
        con.Close();
        Console.ReadLine();
    }
}
```

```

    }
    catch (Exception hata)
    {
        Console.WriteLine(hata.Message);
        Console.ReadLine();
    }
}

```

Uygulamamızı çalıştırdığımızda ilişkili tablolara ait verilerin ekrana yazdırıldığını görürüz.



Şekil 4. MARS Etkisi sayesinde içiçe çalışan SqlDataReader' lar.

Örneğimizde, SqlDataReader nesnelerini kullanarak Yukon sunucusu üzerinde yer alan ilişkili iki tabloya ait verileri console uygulamasından ekrana yazdırmaktayız. dr1 isimli SqlDataReader nesnesi yardımıyla DepartmanSinif isimli master tablodan satırları bir while döngüsü ile okumaktayız. Bu döngünün içerisinde ceeyran eden olaylar ise bizim için MARS etkisinin gücünü bir kere daha göstermektedir. Nitekim, dr1 nesnesi ile master tablodan her bir satır okunduğunda bu satıra ait olan PersonelID alanının değeri, başka bir komuta parametre olarak gitmekte ve en önemlisi de bu komutu bir SqlDataReader nesnesi, diğeri halen daha açıkken yürüterek uygulama ortamına sonuç kümesini döndürmektedir. Aynı kod mantığını Ado.Net 1.1 verisyonunda Sql 2000 sunucusu üzerinde gerçekleştirilmeye çalıştığımız aşağıdaki örneği göz önüne aldığımızda ise;

```

try
{
    SqlConnection con = new SqlConnection("data source=localhost;initial catalog=Northwind;integrated security=SSPI");

    SqlCommand cmd1 = new SqlCommand("SELECT * FROM Orders", con);
    SqlCommand cmd2 = new SqlCommand("SELECT * FROM [Order Details] WHERE OrderID=@OrderID", con);
    cmd2.Parameters.Add("@OrderID", SqlDbType.Int);

    SqlDataReader dr1;

```

```

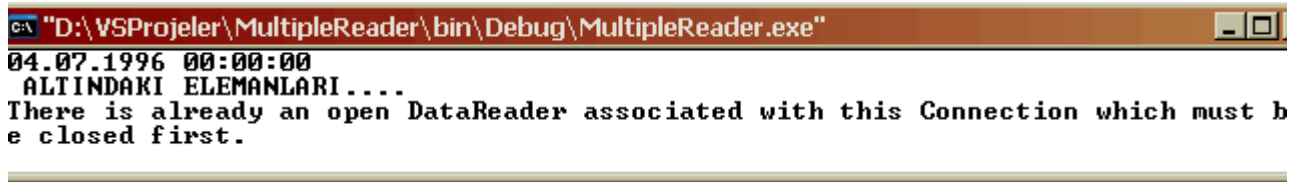
SqlDataReader dr2;

con.Open();
dr1 = cmd1.ExecuteReader();
while (dr1.Read())
{
    Console.WriteLine(dr1["OrderDate"]);
    Console.WriteLine(" ALTINDAKI ELEMENLARI....");
    cmd2.Parameters["@OrderID"].Value = dr1["OrderID"];
    dr2 = cmd2.ExecuteReader();
    while (dr2.Read())
    {
        Console.WriteLine(dr2["ProductID"]+" "+dr2["UnitPrice"]+" "+dr2["Quantity"]);
    }
    dr2.Close();
    Console.WriteLine("-----");
}

con.Close();
Console.ReadLine();
}
catch (Exception hata)
{
    Console.WriteLine(hata.Message);
    Console.ReadLine();
}

```

Bu kez Ado.Net 1.1' in mimarisi bu tarz bir işleme izin vermeyecek ve while döngüsü içinden elde edilen ilk tablo satırından sonra, ikinci SqlDataReader nesnesi yürütülmeye çalışıldığında, halen çalışmakta olan SqlDataReader' ın işlemini sonlandırması gerektiğini söyleyecektir.



Şekil 5. Ado.Net 1.1'deki durum.

Görüldüğü gibi, MARS yapısı var olan sınıfların kabiliyetlerini arttırarak aynı açık bağlantı üzerinden birden fazla sayıda kayıt kümesine erişebilme imkanına sahip olmamızı sağlamıştır. İşin güzel yanı MARS etkisini uygulamalarımıza yansıtılabilmek için, sınıflarımıza has çeşitli ayarlamalar yapamamıza gerek olmayışdır. Aynen Ado.Net 1.1 uygulamaları yazar gibi yazabiliriz kodlarmızı.

Elbetteki bu mimari değişiklikler sayesinde ADO.NET sınıflarının imkan ve kabiliyetleri önemli derecede artmıştır. ADO.NET 2.0 ile gelen tek yenilik MARS değildir. Bunun yanında, aynı anda birden fazla SQL komutunu çalıştırmamıza yarayan asenkron komut yürütme teknikleride eklenmiştir. Bu konuyuda bir sonraki makalemizde incelemeye çalışacağız. Hepinize mutlu günler dilerim.

ADO.NET 2.0 ve SQL Komutlarını Asenkron Olarak Yürütmek – 1

Değerli Okurlarım Merhabalar,

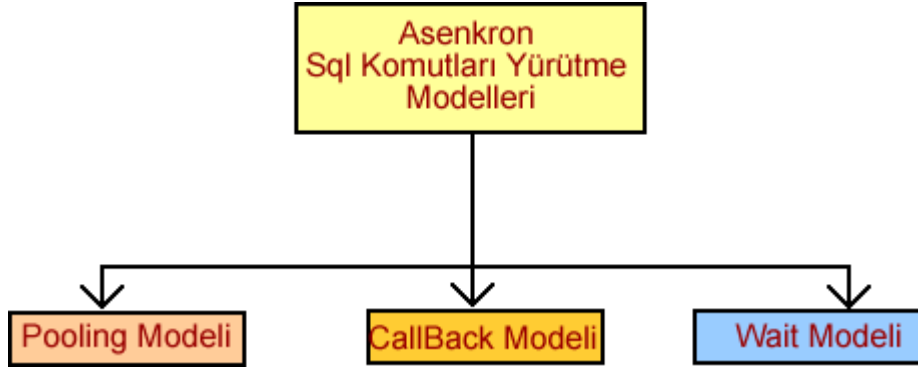
Bir önceki makalemizde MARS etkisini incelemiş ve aynı bağlantı üzerinden birden fazla sayıda sonuç kümesine nasıl erişebileceğimizi görmüştük. Her ne kadar, aynı anda birden fazla sonuç kümesine erişilsekte, halen daha MARS modeli, SQL komutları ile eş zamanlı çalışan kodlar ve asenkron yürütülebilen diğer SQL komutları için yeterli değildir. ADO.NET 2.0 ile, SQL komutlarını asenkron olarak yürütebileceğimiz bir takım yeni üyeler gelmektedir. Bu üyeler sayesinde, SQL komutlarını asenkron olarak çalıştırabilir ve hatta, diğer kod satırlarında eş zamanlı olarak işleyebilmesini sağlayabiliriz. Bu işleri gerçekleştirebilecek üyeler şu an için sadece `SqlClient` sınıfında yer almaktadır. Ancak .NET Framework 2.0' in final sürümünde bu üyelerin, diğer ADO.NET isim alanlarında yerleştirileceklerini düşünüyorum.

Asenkron komut yürütmenin mantığı, asenkron metodların çalıştırılmasına çok benzer. Nitekim burada da, başlatılan bir prosesi kontrol etmemize imkan sağlayan `IAsyncResult` arayüzüne ait bir nesne örneği anahtar görevini oynamaktadır. Asenkron SQL komutlarını yürütmek için, `SqlCommand` sınıfına aşağıdaki tabloda yer alan altı adet yeni metod ilave edilmiştir.

SqlCommand sınıfı için Asenkron komut yürütme metodları
<code>BeginExecuteNonQuery</code>
<code>BeginExecuteReader</code>
<code>BeginExecuteXmlReader</code>
<code>EndExecuteNonQuery</code>
<code>EndExecuteReader</code>
<code>EndExecuteXmlReader</code>

`Begin` ile başlayan tüm metodlar, geriye `IAsyncResult` arayüzü tipinden bir nesne örneği döndürürler. Programın çalışması esnasında, bu tipe ait özellikler kullanılarak çalışan SQL komutlarına ait proseslerin durumları kontrol edilebilir ve tamamlanıp tamamlanmadıkları öğrenilir. Elbetteki, çalışması tamamlanan bir SQL komut metodunun geriye sonuçları döndürebilmesi için, `End` ile başlayan uygun metodun çalıştırılması ve bu metoda ilgili prosese ait `IAsyncResult` nesne örneğinin parametre olarak gönderilmesi

gerekmektedir. Temel olarak bu, Asenkron Sql Komutlarının çalıştırılmasının ana mantığıdır. Bununla birlikte ADO.NET 2.0, asenkron sql komutlarının yürütülebilmesi için 3 değişik model sunmaktadır.



Şekil 1. Asenkron Yürütme Modelleri.

Biz bugünkü makalemizde, Polling Modelini incelemeye çalışacağız. Bu modelde genellikle, `IAsyncResult` tipinden nesne örneği ile sahip olunan prosesin tamamlanıp tamamlanmadığı sürekli olarak kontrol edilir. Yani, `Begin` ile başlayan herhangi bir metoddan sonra bu metodun içerdiği Sql komutu yürütülürken, elde edilen `IAsyncResult` tipi nesne örneğine ait `IsCompleted` özelliğine bakılarak prosesin tamamlanıp tamamlanmadığı araştırılır. Bu araştırma işlemi sürekli tekrar ettiğinden çalıştırılan sql komutları tamamlanıncaya kadar, bu aralıklarda başka işlemler gerçekleştirilebilir veya başka sql komutları yürütülebilir. Diğer taraftan bu kontrol işlemi zorunlu değildir. Nitekim biz bu kontrolü yapmasakta, `Begin` ve `End` komutları arasındaki blokların çalıştırılması sağlanabilir. Ancak burada ana fikir, uzun sürebilecek sql komutlarının asenkron olarak çalışmaları halinde zaman kaybının önüne geçebilmek, uygulamanın işleyişinin kesilmesini önlemek ve bu zaman aralığında başka kodları ve başka kullanıcı aktivitelerini başarılı bir şekilde icra edebilmektir. Şimdi dilerseniz Polling modelinin en basit haliyle uygulanışını incelemek amacıyla, Visual Studio.Net 2005 ortamında aşağıdaki kodlara sahip basit bir Console uygulaması geliştirelim.

```
#region Using directives
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;
```

```
#endregion
```

```
namespace PollingModeli
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
```



```

try
{
    SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI;Asynchronous Processing=true");
    SqlCommand cmd = new SqlCommand("UPDATE Production.Product SET ListPrice=ListPrice+100", con);
    con.Open();
    IAsyncResult res = cmd.BeginExecuteNonQuery();

    int i = 0;
    while (!res.IsCompleted)
    {
        Console.WriteLine("İŞLEM DEVAM EDİYOR "+i);
        i += 1;
    }

    int sonuc = cmd.EndExecuteNonQuery(res);
    Console.WriteLine(sonuc + " SATIR GÜNCELLENDİ...");
    Console.ReadLine();
    con.Close();
}
catch (Exception hata)
{
    Console.WriteLine(hata.Message);
    Console.ReadLine();
}
}
}

```

Şimdi kodlarımızda ne yaptığımıza yakında bakalım. İlk olarak, aşağıdaki satır ile yeni bir SqlConnection nesnesi yaratıyoruz.

```

SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI;Asynchronous Processing=true");

```

Burada, standart Sql Connection ifademizdekilerinin aksine, ekstradan bir terim kullandık. Asynchronous Processing terimi, açılan bağlantının asenkron erişimlere izin verip vermeyeceğini belirler. Varsayılan değeri false' tur ve Asenkron komut yürütmelerine izin vermez. Bu nedenle burada true değerini açıkça atıyoruz. Eğer bu bağlantı özelliğini belirtmesek, çalışma zamanında aşağıdaki istisnayı alırız.



Şekil 2. Bağlantının Asenkron yürütmelere izin vermemesi halinde.

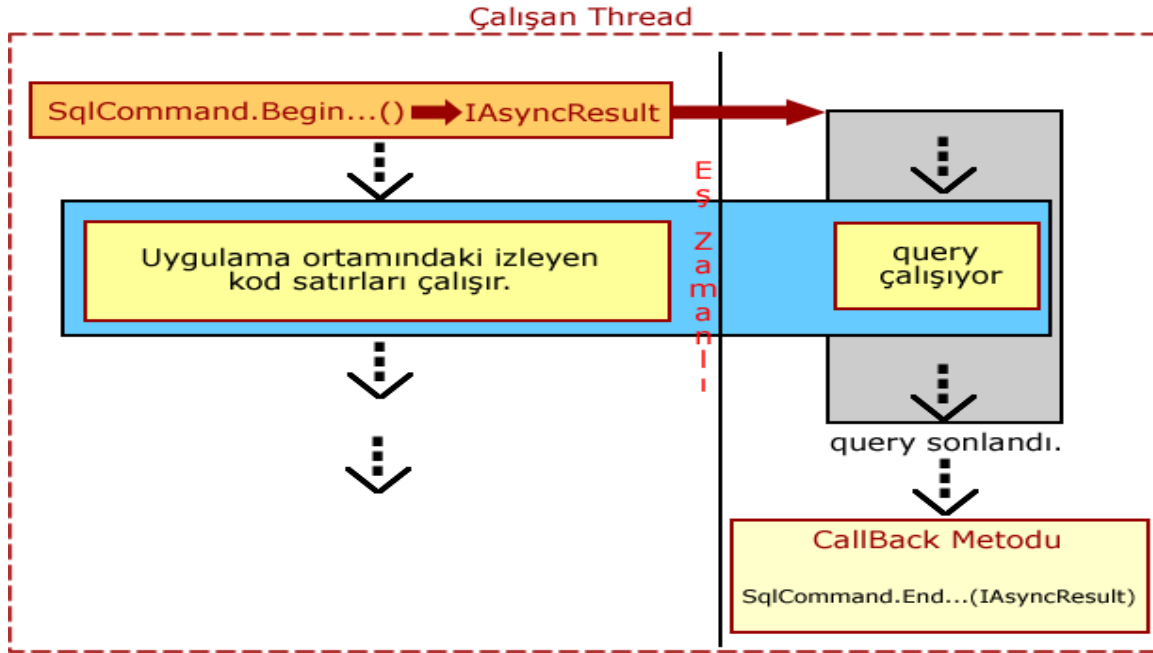
Daha sonra ise, SqlCommand nesnemizi oluşturuyoruz. SqlCommand nesnemiz, AdventureWorks veritabanında yer alan Product isimli tablodaki ListPrice değerini 100 birim arttıran bir Update sorgusuna sahip. Bu tablo Yukon kurulduğunda 504 satır veri içermektedir. Haliyle biraz zaman alabilecek bir güncelleme işlemi söz konusu. Ancak biz istiyoruz ki, bu güncelleme komutu yürütülürken uygulama ortamında başka şeylerde yapabilelim. En azından uygulama bloke olmadan basit bir takım işlemleri gerçekleştirebilme amacındayız. Bu amaçla ilk olarak sql komutumuzu BeginExecuteNonQuery metodu ile çalıştırıyoruz ve hemen ortama bir IAsyncResult tipi nesnenin dönmesini sağlıyoruz.

IAsyncResult res = cmd.BeginExecuteNonQuery();

Şu aşamada, çalışan komuta ait proses res isimli IAsyncResult tipinden nesne örneğinin sorumluluğu altında. Biz bu örneğimizde polling modelini kullandığımız için, eş zamanlı olarak çalıştırmak istediğimiz kodlarımızı bir while döngüsü içerisine alabiliriz. Bu while döngüsü her bir iterasyonunda, IAsyncResult nesnesinin o an sahip olduğu prosesin tamamlanıp tamamlanmadığını kontrol ediyor olacaktır. Bunun içinde, IsCompleted özelliğinin değerine bakılıyor. Eğer dönen değer false ise, işlemin tamamlanmadığı anlaşılıyor ve döngü içindeki kodlar yürütülüyor.

```
while (!res.IsCompleted)
{
    Console.WriteLine("İŞLEM DEVAM EDİYOR "+i);
    i += 1;
}
```

Sql komutunun yürütülmesi tamamlandığında, IAsyncResult nesnemizin IsCompleted özelliği true değerini döndürecektir. Dolayısıyla döngüden çıkmış olunacak ve bir sonraki satıra gelinecektir. Gerçekleştirdiğimiz asenkron işleyişi aşağıdaki şekil ile kafamızda daha kolay canlandırabileceğimizi düşünüyorum.

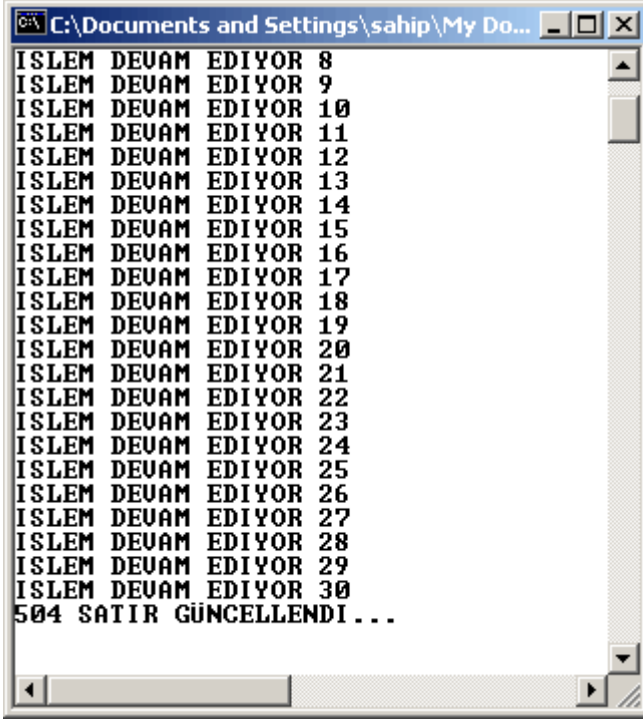


Şekil 3. Polling Modelinde İşleyiş Tarzı.

Tabiki, döngü içerisinde yer alan işlemlerin daha uzun sürede gerçekleşmesi gibi bir durum ilede karşılaşabiliriz. Böyle bir durumda, IAsyncResult nesnesine ait proses tamamlanmış olsa bile, diğer işlemler devam ettiği için bu işlemler sonlanana kadar uygulama donacaktır.

```
int sonuc = cmd.EndExecuteNonQuery(res);
```

Bu satırda ise, EndExecuteNonQuery metoduna yütülen sql komutuna ait prosesi temsil eden IAsyncResult nesnemiz parametre olarak gönderiliyor. Böylece, tamamlanan proses sonucu elde edilen değer (ExecuteNonQuery' nin integer bir değer döndürdüğünü, yani sql sorusundan etkilenen satır sayısını verdiğini belirtelim.) uygulama ortamına aktarılıyor. Uygulamamızı çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 4. Uygulamanın çalışması sonucu.

Elbetteki, bu örneğimizde olduğu gibi bir while döngüsünü kullanmak zorunda değiliz. Sonuç olarak, `IAsyncResult` nesnesi, `End` metodları ile ortama iade edilinceye kadar yer alan tüm satırlar çalışacaktır. Diğer taraftan, aynı anda birden fazla sql komutunun asenkron olarak çalıştırılmasında sağlayabiliriz. Dilerseniz polling modelinin, birden fazla `Sql` komutunun yürütülebilmesi için nasıl kullanılabileceğini bir örnek ile inceleyelim. Bu amaçla, Console uygulamamızdaki kodlarımızı aşağıdaki hale getirelim.

```
try
{
    /* Bağlantımızı oluştururken Asynchronous Processing özelliğine true değerini vermeyi ihmal etmiyoruz.*/
    SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI;Asynchronous Processing=true");

    /* SqlCommand nesnelerimizi oluşturuyoruz. İlki bir tablo üzerinde güncelleme yaparken, ikincisi bir View nesnesinde veri
    çekiyor.*/
    SqlCommand cmd = new SqlCommand("UPDATE Production.Product SET ListPrice=ListPrice+10", con);
    SqlCommand cmd2 = new SqlCommand("SELECT * FROM Sales.vStore", con);
    con.Open(); // Bağlantımızı açıyoruz.

    IAsyncResult res = cmd.BeginExecuteNonQuery();
    IAsyncResult res2 = cmd2.BeginExecuteReader();

    //Bu aralıktaki satırlar bloke olmadan çalışır.
    /* Döngü her iki sql komutuda tamamlanıncaya kadar çalışacaktır. Eğer bunlardan her hangibir önce biterseki öyle olacaktır,
```

diğeride sonlanıncaya kadar döngü içerisindeki kodlar işletilmeye devam eder.*/

```
while ((!res2.IsCompleted)||(!res.IsCompleted))
{
    /* Eğer Update komutu tamamlanmış ise, o anki milisaniye ile birlikte ekrana RES BITTI yazdırır. Eğer işlem
    tamamlanmamışsa, RES ISLEME DEVAM EDİYOR tekstini ve o anki milisaniyeyi yazar.*/
    if (res.IsCompleted)
    {
        Console.WriteLine("RES BITTI ");
        Console.WriteLine(DateTime.Now.TimeOfDay.Milliseconds.ToString());
    }
    else
    {
        Console.WriteLine("RES ISLEME DEVAM EDİYOR " + DateTime.Now.TimeOfDay.Milliseconds.ToString());
    }
    /* Eğer Select komutu tamamlanmış ise, o anki milisaniye ile birlikte ekrana RES2 BITTI yazdırır. Eğer işlem
    tamamlanmamışsa, RES2 ISLEME DEVAM EDİYOR tekstini ve o anki milisaniyeyi yazar.*/
    if (res2.IsCompleted)
    {
        Console.WriteLine("RES2 BITTI ");
        Console.WriteLine(DateTime.Now.TimeOfDay.Milliseconds.ToString());
    }
    else
    {
        Console.WriteLine("RES2 ISLEME DEVAM EDİYOR " + DateTime.Now.TimeOfDay.Milliseconds.ToString());
    }
}

/* Update sorgusunun sonuçlarını ilgil IAsyncResult nesnesini parametre olarak vererek integer bir değışkene atıyoruz. Burada
EndExecuteNonQuery metodu aynen ExecuteNonQuery metodunda olduđu gibi geriye , sorgudan etkilenen satır sayısını
döndürecektir. */
```

```
int sonuc = cmd.EndExecuteNonQuery(res);
Console.WriteLine(sonuc + " SATIR GÜNCELLENDİ...");
```

/* Select sorgusu ile elde edilen kayıt kümesini ortama almak için, EndExecuteReader metoduna res2 isimli IAsyncResult nesne örneğini parametre olarak gönderiyoruz ve bu kümedeki ilk satıra ait 0ncı ve 1nci alanların değerlerini ekrana yazdırıyoruz.*/

```
SqlDataReader dr = cmd2.EndExecuteReader(res2);
dr.Read();
Console.WriteLine(dr[0] + " " + dr[1]);
dr.Close(); // SqlDataReader nesnemizi kapatıyoruz.
Console.ReadLine();
```

```
con.Close(); // Bağlantımızı kapatıyoruz.  
}  
catch (Exception hata)  
{  
    Console.WriteLine(hata.Message);  
    Console.ReadLine();  
}
```

Bu örnekte, iki sql komutunu asenkron olarak çalıştırmaktayız. Yürütülecek her iki komut içinde birer IAsyncResult tipinden nesne örneği oluşturuyoruz. Bu andan itibaren, uygulamada farklı işlemler yapabiliriz. Biz burada, while döngüsünü her iki IAsyncResult nesne örneğinin temsil ettiği proseslerde çalışan sql komutları sonlanıncaya kadar çalıştırmaktayız. Yürütülen komutlar işleyişlerini bitirdiklerinde, kullandıkları IAsyncResult nesne örnekleri yardımıyla asıl sonuçları ortama alıyoruz. Sonuç itibarıyla, çalışan sql komutları asenkron olarak yürütülmekte ve çalışmaları esnasında programın bloke olması önlenmektedir. Aşağıdaki ekran görüntüsünden de dikkat edeceğimiz gibi, daha kısa süren UPDATE işlemi tamamlandıktan sonra, diğer komut bitene dek döngü devam etmiştir.

[illegible]

Şekil 5. Uygulamanın çalışması sonucu.

Bu makalemizde Asenkron Sql Komutların Yürütülmesi konusuna kısaca değinerek kullanabileceğimiz tekniklerden birisi olan Polling Modelini inceledik. Polling modeli aslında, çok büyük boyutlu sorgular içeren sql komutlarının asenkron olarak yürütülmesinde çok faydalı bir teknik değildir. Bunun yerine daha gelişmiş olan diğer modellerden Callback Modeli veya Wait Modeli yararlanılabilir. Bir sonraki makalemizde, Callback modelini incelemeye çalışacağız. Tekrar görüşünceye dek hepinize mutlu günler dilerim.

Ado.Net 2.0 ve Sql Komutlarını Asenkron Olarak Yürütmek - 2

Değerli Okurlarım Merhabalar,

Hatırlayacağınız gibi bir önceki makalemizde, sql komutlarının asenkron olarak yürütülmesi için kullanılan tekniklerden birisi olan polling modelini incelemiştik. Polling modeli basit olmakla birlikte, iş yükü fazla olan hacimli sql komutlarının asenkron olarak çalıştırılmasında çok fazla tercih edilmemelidir. Bu tip sorguların yer aldığı asenkron yürütmelerde, Callback veya Wait modellerini kullanmak verimliliği arttırıcı etkenlerdir. Bu makalemizde Callback modelini kısaca incelemeye çalışacağız.

CallBack modeli anafikir olarak, asenkron olarak çalışan sql komutlarının işleyişlerinin sona erdiği noktalarda yürürlüğe giren metodları bünyesinde barındıran bir tekniktir. Bu tekniğe göre, asenkron olarak yürütülecek sql komutlarını taşıyan SqlCommand nesneleri yine bilinen Begin... metodları ile çalıştırılırlar. Ancak bu kez, SqlCommand nesnesine ait Begin metodlarının aşağıdaki tabloda belirtilen aşırı yüklenmiş versiyonları kullanılır.

CallBack modelinde kullanılan SqlCommand.Begin... metodları

```
public IAsyncResult BeginExecuteNonQuery ( AsyncCallback callback, object stateObjcet );
```

```
public IAsyncResult BeginExecuteReader ( AsyncCallback callback, object stateObjcet );
```

```
public IAsyncResult BeginExecuteXmlReader ( AsyncCallback callback, object stateObjcet );
```

Burada görüldüğü gibi her üç metod da, iki parametre almaktadır. İlk parametre AsyncCallback temsilcisi tipindendir. Bu parametre yardımıyla, yürütülecek olan sql komutları tamamlandığında çalıştırılacak olan metod işaret edilir. Bu, static olan, void geri dönüş değerine sahip ve yalnızca IAsyncResult tipinde bir nesne örneğini parametre olarak alan bir metod olmalıdır. Bir başka deyişle Begin metodu ile çalıştırılan sql sorguları sonlandığında hangi metodun çalıştırılacağı buradaki temsilci (delegate) yardımıyla belirlenmiş olunur.

İkinci parametre ise kullanıcı tarafından belirtilebilen object tipinden bir nesnedir. Çoğunlukla, CallBack metodu içine, asenkron olarak çalışan sql komutunun sahibi olan SqlCommand nesnelerini aktarmak amacıyla kullanılmaktadır. Polling modelinde olduğu gibi burada da Begin metodlarının geriye dönüş değerleri, çalışan asenkron sorgudan sorumlu olan IAsyncResult arayüzü tipinden nesne örnekleridir.

Bu kısa açıklamalardan sonra dilerseniz, CallBack modelinin nasıl uygulandığını gösteren basit bir örnek geliştirelim. Bu amaçla Visual Studio.Net 2005' te aşağıdaki kodlara sahip olan bir Console uygulaması oluşturalım.

```
#region Using directives
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;
```

```
#endregion
```

```
namespace CallBackModel
```

```
{
    class Program
    {
        public static void Main(string[] args)
        {
```



```

        SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI;async=true");
        con.Open();
        SqlCommand cmd = new SqlCommand("UPDATE Production.Product SET ListPrice=ListPrice*1.15", con);
        SqlCommand cmd2 = new SqlCommand("SELECT * From Person.Address", con);
        IAsyncResult res = cmd.BeginExecuteNonQuery(new AsyncCallback(UPDATE_OK), cmd);
        IAsyncResult res2 = cmd2.BeginExecuteReader(new AsyncCallback(SELECT_OK), cmd2);
        Console.WriteLine("SORGULAR ÇALIŞIYOR...");
        Console.ReadLine();
        con.Close();
    }

    public static void UPDATE_OK(IAsyncResult r)
    {
        SqlCommand komut = (SqlCommand)r.AsyncState;
        int sonuc=komut.EndExecuteNonQuery(r);
        Console.WriteLine(sonuc + " SATIR GÜNCELLENDİ...");
    }

    public static void SELECT_OK(IAsyncResult r)
    {
        SqlCommand komut = (SqlCommand)r.AsyncState;
        SqlDataReader dr = komut.EndExecuteReader(r);
        dr.Read();
        Console.WriteLine(dr[1] + " " + dr[2]);
    }
}
}

```

Uygulamayı çalıştırmadan önce kısaca neler yaptığımıza daha yakında bakmakta fayda var. Bu örneğimizde, Yukon üzerinde yer alan AdventureWorks veritabanına bağlandık. Amacımız iki sql sorgusunu asenkron olarak çalıştırmak ve bu sorgular işlerken uygulama ortamında izleyen kod satırlarını yürütülebilmesini sağlamak. SqlCommand nesnelerimizde yarattıktan sonra sıra bu komutları ilgili Begin... metodları ile çalıştırmaya geliyor.

```

IAsyncResult res = cmd.BeginExecuteNonQuery(new AsyncCallback(UPDATE_OK), cmd);
IAsyncResult res2 = cmd2.BeginExecuteReader(new AsyncCallback(SELECT_OK), cmd2);

```

Burada bizim için en önemli noktalar parametrelerdir. Her iki metodun ilk parametresi aslında AsyncCallback temsilcisi tipindedir. Nitekim;

new AsyncCallback(UPDATE_OK)

tanımlaması ile aslında UPDATE_OK isimli metodu işaret eden AsyncCallback tipinden bir temsilci tanımlanmış olmaktadır. Bu temsilcinin yaptığı iş sadece, IAsyncResult nesne örneğinin sorumluluğunda olan sql komutlarının yürütülüşü tamamlandığında, derhal çalıştırılacak olan metodun hangi metod olacağına karar vermektir.

İkinci parametremiz ise, Begin metodunun sahibi olan SqlCommand nesnesidir. Bu nesneye, UPDATE_OK metodu içinden erişilebiliriz. Eğer böyle bir erişim söz konusu olmasaydı, Callback metodunun içinden sql sorgusunun sahibi olan SqlCommand nesnesine erişmekte sorunlar yaşayabilirdik. Nitekim Callback metodu dikkat edeceğimiz gibi static olmak zorundadır ve static bir metod içinde static olmayan üyeler erişmekte sorun yaşamaktayızdır. Tüm bunlar AsyncCallback temsilcisinin işaret edebileceği metodun yapısı ile ilgilidir.

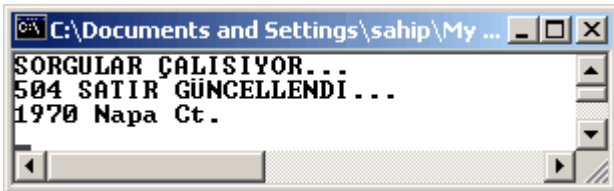
```
public static void UPDATE_OK(IAsyncResult r)
{
    SqlCommand komut = (SqlCommand)r.AsyncState;
    int sonuc=komut.EndExecuteNonQuery(r);
    Console.WriteLine(sonuc + " SATIR GÜNCELLENDİ...");
}
```

Bu metod dikkat edecek olursanız, IAsyncResult tipinden bir nesneyi parametre olarak almaktadır. Buraya geçirilen bu parametre, Begin metodu ile oluşturulan IAsyncResult nesne örneğidir. Dolayısıyla, sql komutunun işleyişi tamamlandığında devreye girecek olan bu metod içinden, IAsyncResult nesne örneği yardımıyla sonuç kümelerini elde edebilmemiz mümkün olabilmektedir. Bir başka deyişle, End... metodunu buraya aktarılan IAsyncResult nesne örneği üzerinden çağırabiliriz. Diğer yandan, sql komutunun sahibi olan SqlCommand nesnesini metod içinde kullanabilmek için,

```
SqlCommand komut = (SqlCommand)r.AsyncState;
```

satırı kullanılmıştır. Bu satırda dikkat ederseniz, IAsyncResult nesnesinin AsyncState özelliğinden yararlanılmıştır. Bu özellik, asenkron yürütme operasyonunda görev olan SqlCommand nesnesini elde etmemizi sağlar. Tabiki özelliğin tanımlanışı gereği, geri dönüş değeri object tipinde olduğundan burada bir cast işlemi uygulanmıştır. Yani object nesneyi SqlCommand tipine çevirmemiz gerekmektedir.

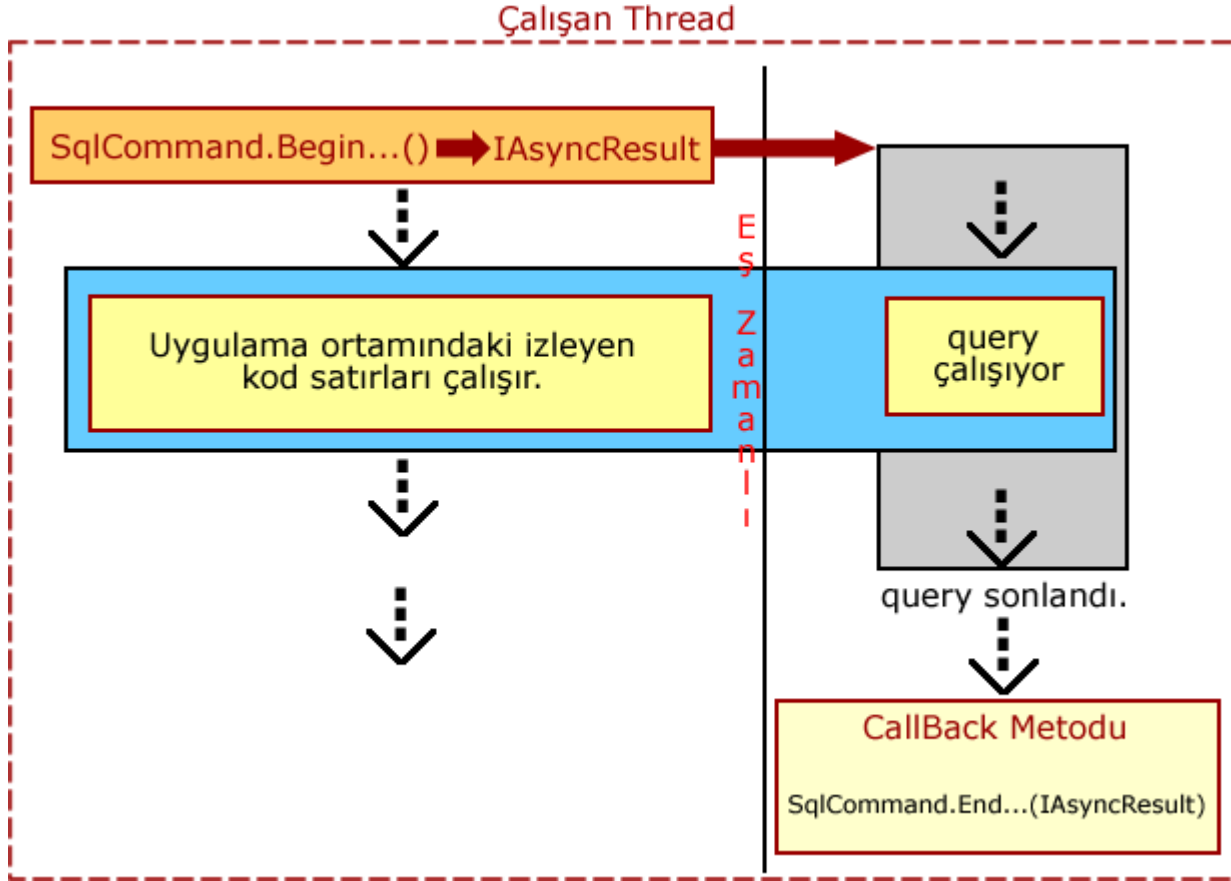
Son adımda ise, EndExecuteNonQuery metodu çağırılarak, sql komutunun sonuçları elde edilmiştir. Bu metod için kullanılan desen yapısı Callback metodu olan SELECT_OK içinde geçerlidir. Uygulamamızı çalıştırdığımızda aşağıdaki ekran görüntüsünü elde ederiz.



Şekil 1. Uygulamanın çalışması sonucu.

Dikkat edecek olursanız, biz Callback tekniğini uygulamaya başladığımız Begin... çağrılarının sonraki kodlar, sql sorgularımızın tamamlanmasından önce çalışmıştır.

Bu modeli, bir önceki makalemizde incelediğimiz polling modeli ile karşılaştırdığımızda ilk göze çarpan, sürekli olarak bir kontrol yapmayışımdır. Hatırlayacağınız gibi polling modelinde, IAsyncResult nesnesinin IsCompleted özelliği ile, çalışan sql sorgularının tamamlanıp tamamlanmadığı kontrol edilmektedir. Bu her ne kadar basit ve fazla zaman almayan sorgular için kullanışlı olsada, daha büyük ölçekli sorgularda, çalışan asenkron prosesin sürekli olarak kontrol edilmeye çalışılması çok anlamlı olmayacak ve en önemlisi zaman kaybında neden olacaktır. Bu nedenle bu tip yoğun sorgularda genellikle Callback veya Wait modellerinden birisi tercih edilir. Callback modelinin anatomisini zihninizde daha iyi canlandırabilmek amacıyla aşağıdaki şekli de göz önüne alabiliriz.



Şekil 2. Callback modelinin anatomisi.

Görüldüğü gibi aslında, asenkron olarak çalışacak sql sorguları tamamlandığında devreye giren Callback metodları, uygulama ortamındaki satırlardan tamamen bağımsız olarak çalışan yapılardır. Sql komutları sonlandığında anında devreye girerek sonuçların elde edilebilmesini sağlayan ve hatta başka işlemlerin gerçekleştirilebileceği kod bloklarını kapsüllememize imkan vermektedirler. Callback yapısının biraz daha gelişmiş bir versiyonu olan Wait modelide asenkron sql komutları yürütülmesinde kullanılan tekniklerdendir. Bir sonraki makalemizde bu konuya değinmeye çalışacağız. Tekrar görüşünceye dek hepinize mutlu günler dilerim.

Xml Web Servislerine Giriş – 1

Bu makalemizde, kısaca bir XML Web Servisinin ne olduğuna, ne işe yaradığına değinecek ve basit bir Xml Web Servisinin notepad ile nasıl oluşturulabileceğini incelemeye çalışacağız.

Bir Web Servisi, uzak istemcilerin başvuruda bulunduğu çeşitli işlevsel metod çağırımlarını bardındırdan, çok yönlü ve merkezileştirilmiş bir ünitelerdir. Bir web servisi, çok sayıda istemci tarafından erişilebilen bir yapıya sahiptir. Onu diğer dağıtık nesne

modellerinden farklı kılan sahip olduğu alt yapı sistemi sayesinde, platform bağımsız uygulanabilirliği sağlamasıdır. Web servislerinin geliştirilmesinde en büyük etken, özellikle bir merkezdeki uygulamalar üzerinde, ortak amaçları gerçekleştiren işlevselliklere sahip nesnelerin, geliştirildikleri ağın iletişim protokolü gibi kısıtlamaların varlığıdır.

Bir web servisi, standart olarak HTML iletişim protokolü üzerinden veri alışverişine izin veren bir yapıdadır. HTML tabanlı bu sistemin bilgi otobanı XML temelleri üzerine dayandırılmıştır. XML' in bizlere sağladığı esneklik, kolay geliştirilebilirlik özelliklerinin yanı sıra, sağlamış olduğu global standartlar, platform bağımsız veri transferi kavramını dahada geliştirmiştir. Web servislerinin kullanılmasında yatan en büyük kavram xml tabanlı veri akışının belirli standartlar dahilinde gerçekleştirilmesidir. Bu, web servislerinin platform bağımsız olarak herhangi bir ateş duvarına (Firewall) yakalanmadan istemciler ile haberleşebilmesini sağlar.

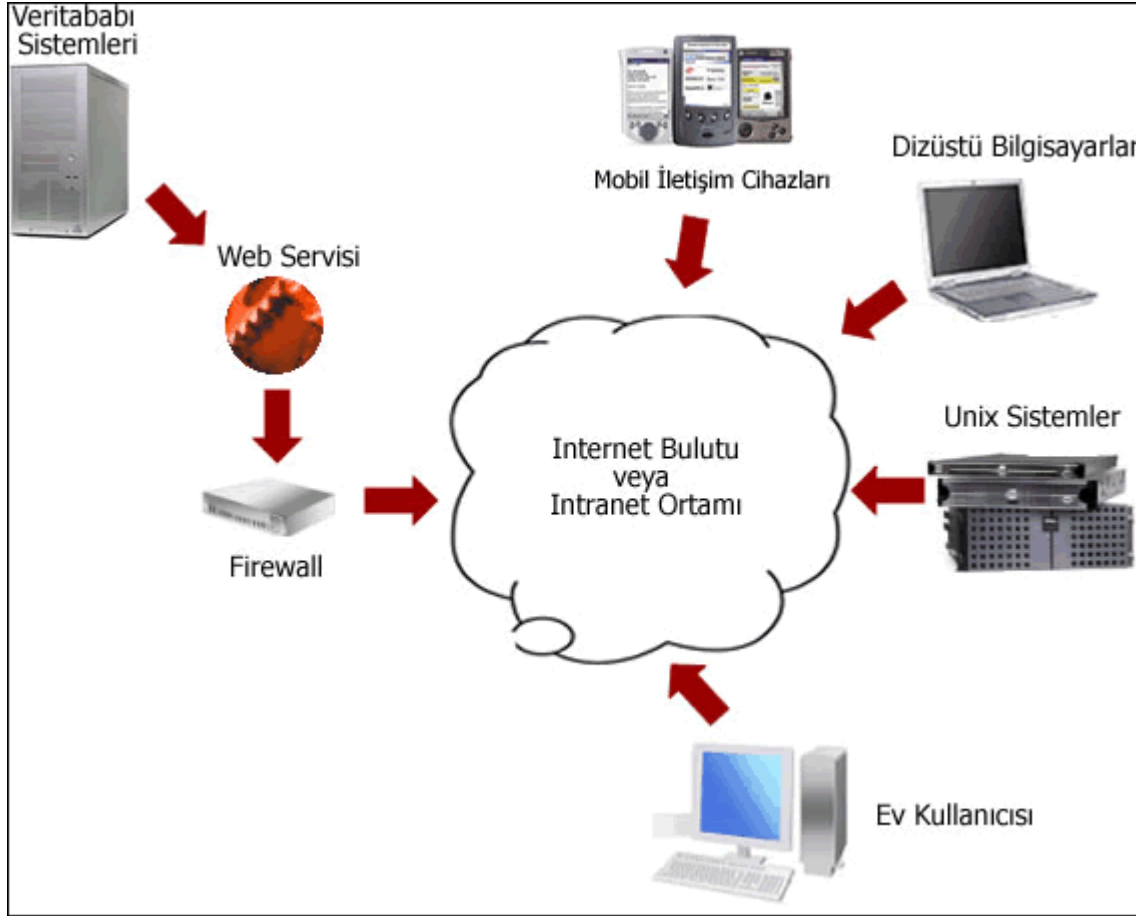


Şekil 1. XML Web Servisleri Neye İfade Eder?

Bir web servisi, tek başına bir anlam ifade etmez. Web servisini kullanan istemcilerin de olması gerekir. İstemciler internet ortamında olabileceği gibi, çalıştığımız şirketin network sisteminde yada evimizdeki makinenin yerel sunucusu üzerinde olabilir. Bir istemci, bir web servisini kullanmak istediğinde tek yapması gereken, bu web servisi ile konuşabilecek ortak bir takım standartları uygulamaktır. XML tabanlı bu standartlar sayesinde istemciler, web servisine ulaşabilir, bu servis üzerinden metodlar çağırabilir, bu metodlara parametreler gönderebilir ve metodlardan dönen değerleri örneğin veri kümelerini elde edebilir.

Web servislerinin kullanımına verilebilecek en güzel örnek, hava durumuna ilişkin bilgilerinin en güncel halleriyle, çeşitli platformlarda çalışan istemcilere sunulduğu bir sistem olabilir. Hava durumuna ilişkin çeşitli bilgileri bir veri kümesi halinde tedarik eden merkezi web servisine istemciler, bir web sayfasından, bir windows yada java uygulamasından, bir mobil uygulamadan veya başka bir platformdan kolayca erişebilir. Web servisleri merkezi uygulamalar olduklarından, verilerdeki değişiklikler bu servisleri

kullanan tüm uygulamalar için de eş zamanlı ve eş güncellikte olacaktır. Web servisleri ve bu servisleri kullanan istemciler arasındaki ilişkiler, yüzeysel olarak bakıldığında aşağıdaki şekilde görüldüğü gibi değerlendirilebilir.



Şekil 2. En basit haliyle Xml Web Servislerinin hayatımızdaki yeri.

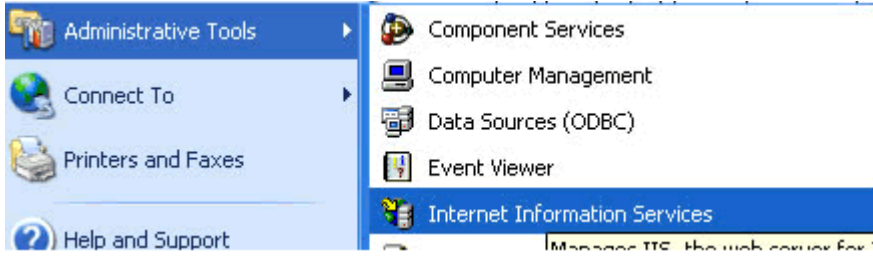
Web servislerinin, onları kullanan istemciler ile arasındaki ilişki, şüphesiz ki bu şekilde görüldüğü kadar basit değildir. Herşeyden önce, web servislerini kullanacak istemciler ile arada kurulacak ilişkinin belli standartlara dayandırılması gerekir. Her ne kadar, web servisleri HTML üzerinden gidecek XML veri parçalarını kullanıyor olsada bunların, istemcilerin işleyebileceği ve anlayabileceği bir hale getirilmeleri gerekir. Bu noktada devreye Web Servisleri için önemli ve gerekli temellerden birisi olan SOAP (Simple Object Access Protocol - Basit Nesne Erişim Antlaşması) girer.

Bir istemci, kullanacağı web servisine ait bir takım bilgilere sahip olmak zorundadır. İstemci bu bilgileri kullanarak web servisinden SOAP protokolüne uygun olarak hazırlanan XML mesajını gönderir. Kodlanarak (Encoding) gönderilen bu mesaj, Web Servisi tarafından çözülür (Decoding) , gerekli parametreler ve metod çağırımı bilgileri eşliğinde bir takım işlemler gerçekleştirir. Bu işlemler sonrasında Web Servisi, istemciye döndüreceği cevap bilgileri için yine SOAP protokolüne uygun XML mesajlarını oluşturur. Bu mesajlar HTTP üzerinden istemci uygulamaya ulaşır, burada çözülür ve değerlendirilir.

Elbette şekildeki senaryoda yer alan istemci sistemler bunlar ile sınırlı değildir. Çok çeşitli platformlar da web servislerini kullanabilir. Web servisinin önünde çalışan bir Firewall (Ateş Duvarı) olması bilgilerin kolayca taşınabilmesini engellemez. Çünkü mesajlar, istemciler ve web servisleri arasında XML tabanlı bilgi parçacıkları şeklinde taşınmaktadır.

İstemcilerin kullanacakları web servisindeki bilgileri önceden bilmeleri gerekir. WSDL (Web Services Description Language - Web Servisleri Tanımlama Dili) bu noktada devreye giren bir diğer önemli unsurdur. İstemci uygulamalar WSDL yardımıyla, kullanacakları web servisine ait bilgileri önceden tedarik ederler. Bu istemcinin web servisi üzerindeki bir web metodunun varlığından haberdar olması, onu nasıl kullanacağını bilmesi anlamına gelmektedir. Web servislerinin mimarisini daha derin ve detaylı bir şekilde incelemeden önce, ilk Web Servisi uygulamamızı yazmaya başlıyoruz.

Bir Web Servisini oluşturmak için, Notepad gibi basit bir metin editorunu kullanabileceğimiz gibi, Visual Studio.NET gibi ileri seviyede bir yazılım geliştirme platformunda kullanabiliriz. İlk önce Notepad üzerinden bir web servisinin nasıl yazılacağını göreceğiz. Bir web servisi her şeyden önce, intranet veya internet üzerinde yer alan bir sunucuda konuşturılmalıdır. Yerel bir makinede bu iş için, IIS (Internet Information Services) kullanılabilir. Bu nedenle ilk olarak, web servisimizi barındıracak sanal bir klasör oluşturmakla işe başlamalıyız. Windows XP işletim sistemine sahip bir bilgisayarda, IIS altında sanal klasörümüzü oluşturabilmek için Start menüsü, Administrative Tool, Internet Information Services kısmına girelim.



Şekil 3. IIS

Ardından Default Web Site (yada web sunucusunun adı) kısmında sağ menü tuşuna basıp New kısmından Virtual Directory' yi seçelim.



Şekil 4. Virtual Directory.

Karışımıza çıkacak olan sihirbazda Alias kısmına Geometri girelim. Klasörümüzü ise, C:\Inetpub\wwwroot\Geometri olarak oluşturalım. Buradaki adımları tamamladıktan sonra, web servisimizi bu klasör altında oluşturabiliriz. Web servisleri, asmx uzantılı dosyalar olarak oluşturulurlar. Bu nedenle, asmx uzantılı dosyamızı, yerel web sunucumuzda oluşturduğumuz sanal klasörün işaret ettiği fiziki klasörde aşağıdaki kodlar ile hazırlayalım.

```
<% @ WebService Language="C#" CodeBehind="GeoMat.asmx.cs" class="GeoWebServis.TemelIsler" %>
```

Hazırladığımız bu dosyayı, GeoMat.asmx uzantısı ile kaydedelim. Kodlarda görüldüğü gibi, web servisimizin asıl işlevselliğini, GeoMat.asmx.cs isimli Code-Behind dosyasında gerçekleştireceğiz. Burada kullanacağımız programlama dilinde Language özelliğine

C# değerini atayarak belirledik. Ancak en önemlisi kaydedilen bu asmx dosyasının bir web servisi olarak değerlendirileceğini belirten WebService anahtar sözcüğünün kullanılmasıdır. Şimdi Code-Behind dosyamızı oluşturalım.

```
using System;
using System.Web;
using System.Web.Services;
```

```
namespace GeoWebServis
{
```

```
    [WebService(Namespace="http://ilk/servis/",Description="Geometrik Hesaplamalar Üzerine Metodlar İçerir. Ucgen, Dortgen gibi şekillere yönelik alan ve çevre hesaplamaları.",Name="Geometrik Hesaplamalar")]
```

```
    public class TemelIsler : System.Web.Services.WebService
```

```
    {
```

```
        private const double pi=3.14;
```

```
        [WebMethod(Description="Daire Alan Hesabı Yapar")]
```

```
        public double DaireAlan( double r)
```

```
        {
```

```
            return (r*r)*pi;
```

```
        }
```

```
        [WebMethod(Description="Daire Çevre Hesabı Yapar.")]
```

```
        public double DaireCevre( double r)
```

```
        {
```

```
            return 2*pi*r;
```

```
        }
```

```
    }
```

```
}
```

Yazdığımız bu dosyayı GeoMat.asmx.cs ismi ile asmx dosyamızın bulunduğu klasöre kayıt edelim. Kodları kısaca incelediğimizde ilk olarak, köşeli parantezler içerisindeki ifadeler dikkatimizi çekmektedir. Bu ifadeler birer nitelik (attribute) olup, web servisindeki sınıf ve metodlara ilişkin bir takım bilgileri, bu servisi kullanacak olan istemcilere sağlarlar.

Örneğin WebService niteliğinde, Namespace özelliği ile, bu servisin tanımını içerecek XML dökümanında kullanılacak Namespace' i belirtmiş oluruz. Description özelliği ile, web servisindeki bu sınıfın neler yaptığını özetleyen kısa bilgileri belirleriz. Name özelliği ilede, bu sınıfa bir isim vermiş oluruz. Bu bilgiler özellikle web servisini keşfettiğimizde (Discovery) , oldukça işe yaramaktadır. WebService niteliğine benzer olarak WebMethod niteliği, izleyen metodun bir web servisi metodu olduğunu belirtmek için kullanılır. Burada iki web servisi metodumuz yer almaktadır. Her birinin ne iş yaptığına dair kısa bilgileri de WebMethod niteliğinin Description özelliği ile belirtebiliriz.

Burada kullandığımız isim alanı ve sınıf adlarının, asmx dosyasında belirttiğimiz class tanımlamalarındaki ile aynı olduğuna dikkat edelim. Bir web servisi yazdığımızda, bu web servisini oluşturacak olan sınıfın, System.Web.Services.WebService isim alanından türetilmesi gerekir. Yazmış olduğumuz bu web servisi temel olarak iki metoda sahiptir ve bu metodların herbiri double türünden birer parametre alarak, yine double türünden sonuçlar üretmekte ve metodun çağırıldığı yere döndürmektedirler. Burada kullanılan parametreler istemci bilgisayarların, web servisindeki ilgili metodları çağırımları sırasında kullanılır. Dönüş değerleri ise, ilgili metodları çağıran istemcilere gönderilir.

Web servisimize ait asmx dosyamızı ve Code-Behind dosyamızı oluşturduktan sonra, Code-Behind dosyamızı dll kütüphanesi olarak derleyip, bin isimli bir klasör içerisine koymalıyız. Bu amaçla komut satırından aşağıdaki komutu vererek, Code-Behind dosyamızı bir sınıf kütüphanesi olacak şekilde csc yardımıyla derliyoruz.

```
csc /target:library GeoMat.asmx.cs
```

Oluşturulan dll dosyasını asmx dosyamızın bulunduğu klasör altındaki bin isimli bir klasör içerisine taşıdıktan sonra web servisimize herhangi bir tarayıcıdan rahatlıkla erişebiliriz. Bunun için, Internet Explorer penceresinde adres satırına

```
http://localhost/Geometri/GeoMat.asmx
```

url bilgisini yazmamız yeterli olacaktır. Bu işlem sonrasında web servisimizin çalışır hali aşağıdaki gibi olacaktır.



Şekil 5. Xml Web Servisinin tarayıcıdan talep edilmesinin sonucu.

Görüldüğü gibi asmx uzantılı dosyamızı kullanarak, geliştirmiş olduğumuz web servisine ait bilgilere eriştik. Burada yazılan bilgilerin, WebService ve WebMethod niteliklerinde belirtmiş olduğumuz değerlerin aynısı olduğu dikkatinizi çekmiştir. Dilersek bu pencerede servisimizi deniyebiliriz. Web servisimizde geliştirdiğimiz metodlara ait linklerden herhangi birisine tıkladığımızda, bu metodu çağırmanın için kullanabileceğimiz bir sayfa ile karşılaşırız.

Address  <http://localhost/Geometri/GeoMat.aspx?op=DaireCevre>

Geometrik Hesaplamalar

Click [here](#) for a complete list of operations.

DaireCevre

Daire Çevre Hesabi Yapar.

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
r:	

Şekil 6. Bir Web Metodunun çağırılması.

Bu ekranda, value ile belirtilen bir metin kutusu olduğuna dikkat edin. Bu metin kutusu metodumuzun dışarıdan aldığı double tipten parametreye istinaden oluşturulmuştur. Buraya bir değer girerek metodun çağırılmasını sağlayabiliriz. Bu durumda metodun çalıştırılması sonucu elde edilecek sonuç(lar) bir XML bilgisi şeklinde elde edilecek ve tarayıcı penceresinde aşağıda olduğu gibi görünecektir.

```
<?xml version="1.0" encoding="utf-8" ?>
<double xmlns="http://ilk/servis/">62.800000000000004</double>
```

Web servisimizin ana sayfasında Service Description isimli bir bağlantı vardır. Bu bağlantıya tıkladığımızda aşağıdaki gibi uzun bir XML bilgisi elde ederiz.

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://ilk/servis/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://ilk/servis/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
- <types>
- <s:schema elementFormDefault="qualified" targetNamespace="http://ilk/servis/">
- <s:element name="DaireAlan">
- <s:complexType>
```

```

- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="r" type="s:double" />
</s:sequence>
</s:complexType>
</s:element>
- <s:element name="DaireAlanResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="DaireAlanResult" type="s:double" />
</s:sequence>
</s:complexType>
</s:element>
- <s:element name="DaireCevre">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="r" type="s:double" />
</s:sequence>
</s:complexType>
</s:element>
- <s:element name="DaireCevreResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="DaireCevreResult" type="s:double" />
</s:sequence>
</s:complexType>
</s:element>
</s:schema>
</types>
- <message name="DaireAlanSoapIn">
  <part name="parameters" element="s0:DaireAlan" />
</message>
- <message name="DaireAlanSoapOut">
  <part name="parameters" element="s0:DaireAlanResponse" />
</message>
- <message name="DaireCevreSoapIn">
  <part name="parameters" element="s0:DaireCevre" />
</message>
- <message name="DaireCevreSoapOut">
  <part name="parameters" element="s0:DaireCevreResponse" />

```

```

</message>
- <portType name="Geometrik_x0020_HesaplamalarSoap">
- <operation name="DaireAlan">
  <documentation>Daire Alan Hesabi Yapar</documentation>
  <input message="s0:DaireAlanSoapIn" />
  <output message="s0:DaireAlanSoapOut" />
</operation>
- <operation name="DaireCevre">
  <documentation>Daire Çevre Hesabi Yapar.</documentation>
  <input message="s0:DaireCevreSoapIn" />
  <output message="s0:DaireCevreSoapOut" />
</operation>
</portType>
- <binding name="Geometrik_x0020_HesaplamalarSoap" type="s0:Geometrik_x0020_HesaplamalarSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <operation name="DaireAlan">
  <soap:operation soapAction="http://ilk/servis/DaireAlan" style="document" />
- <input>
  <soap:body use="literal" />
</input>
- <output>
  <soap:body use="literal" />
</output>
</operation>
- <operation name="DaireCevre">
  <soap:operation soapAction="http://ilk/servis/DaireCevre" style="document" />
- <input>
  <soap:body use="literal" />
</input>
- <output>
  <soap:body use="literal" />
</output>
</operation>
</binding>
- <service name="Geometrik_x0020_Hesaplamalar">
  <documentation>Geometrik Hesaplamalar Üzerine Metodlar İçerir. Ucgen, Dortgen gibi sekillere yönelik alan ve çevre hesaplamalari.</documentation>
- <port name="Geometrik_x0020_HesaplamalarSoap" binding="s0:Geometrik_x0020_HesaplamalarSoap">
  <soap:address location="http://localhost/Geometri/GeoMat.asmx" />

```

```
</port>  
</service>  
</definitions>
```

Burada görüldüğü gibi bir XML belgesine neden ihtiyacımız olabilir? Dikkat edilecek olursa, bu XML belgesinde, Web Servisimize ait bir takım bilgiler yer almaktadır. Örneğin, Metotlara ilişkin parametre bilgileri veya WebService ve WebMethod niteliklerinde belirttiğimiz tanımlamalar gibi. İşte istemci bilgisayarlar bu XML çıktısını kullanarak, iletişim kuracakları web servsileri hakkında bilgi sahibi olurlar. Ancak burada asıl önemli olan nokta, Service Description bağlantısının, aşağıdaki şekilde oluşudur. Bu aslında, bir istemcinin, herhangi bir web servisi hakkındaki bilgilere nasıl ulaşabileceğini göstermektedir.

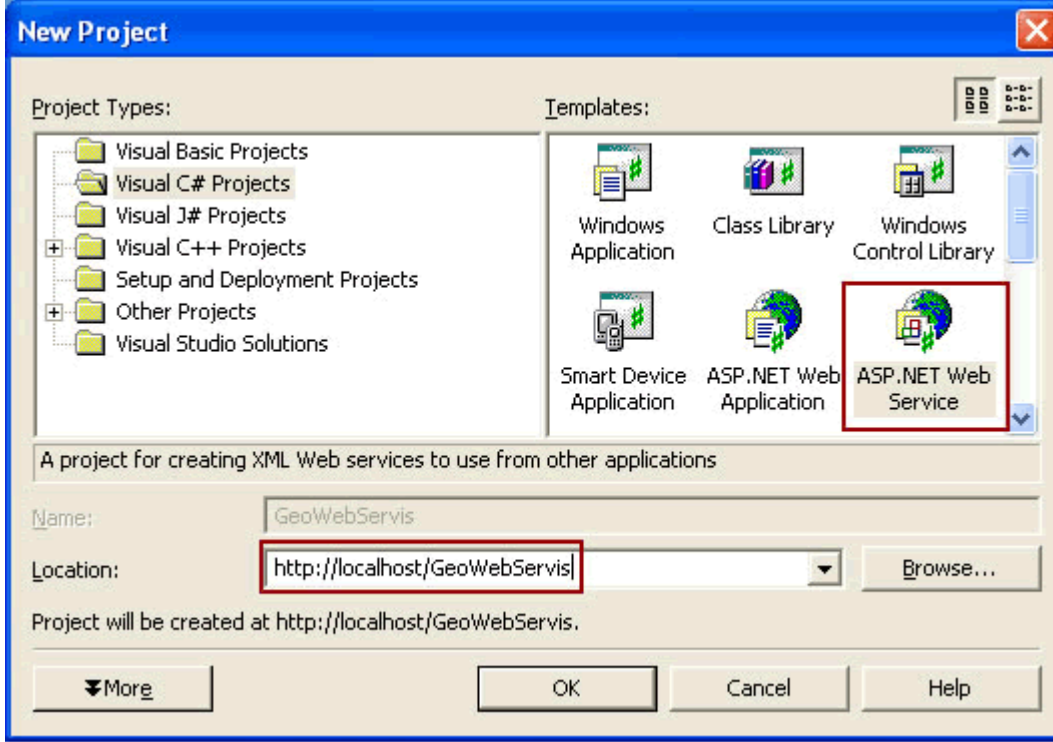
<http://localhost/Geometri/GeoMat.asmx?WSDL>

Bir Web Servisini Visual Studio.NET ortamında geliştirmek, şu ana kadar yaptıklarımızdan pek farklı değildir. Ancak Visual Studio.Net ortamının sağladığı avantajlar nedeniyle çok daha kolaydır. Bir sonraki makalemizde bir web servisinin Visual Studio.Net ortamında nasıl geliştirileceğini ve bu web servisine erişecek bir istemcinin nasıl yazılacağını incelemeye çalışacağız. Hepinize mutlu günler dilerim.

Xml Web Servislerine Giriş – 2

Değerli Okurlarım, Merhabalar.

Bu makalemizde, bir Xml Web Servisinin Visual Studio ile nasıl oluşturulabileceğini ve bir web sayfası üzerinden nasıl çağırılıp kullanılabileceğini incelemeye çalışacağız. Visual Studio.Net ortamında bir web servisi geliştirmek için, ilk olarak New Project bölümünden, ASP.NET Web Service şablonu seçilir. Visual Studio.Net, yerel makinede bu web servisi için gerekli fiziki ve sanal klasörleri, otomatik olarak oluşturacaktır. Notepad editoründe yazdığımız örneğin aynısını, Visual Studio.Net ortamında gerçekleştireceğimizden, proje ismi olarak GeoWebServis' i kullanalım. Bu aynı zamanda web servisimizin varsayılan isim alanı (default namespace) olacaktır.



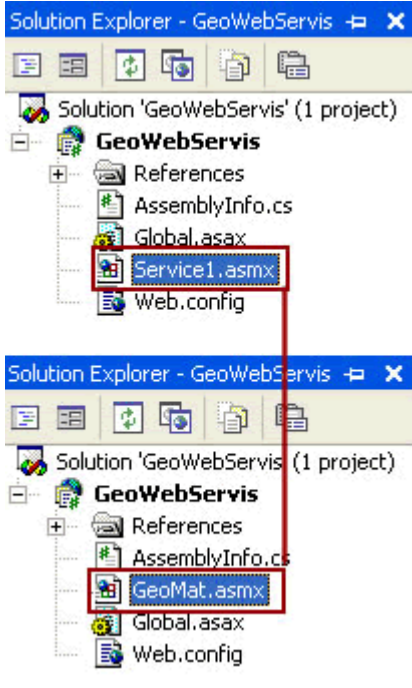
Şekil 1. Yeni bir Web Servis projesinin eklenmesi.

Bu işlemin ardından Visual Studio.Net, web servisimiz için gerekli dosyaları oluşturur. Varsayılan olarak servisimiz, Service1.asmx adını alacaktır. Bununla birlikte, bu servise ait Code-Behind dosyasının kodlarında Visual Studio.Net tarafından otomatik olarak hazırlanır. Visual Studio.Net, servisin kullanımına örnek teşkil edecek bir metodu da yorum satırları halinde sunmaktadır. (HelloWorld() metodu)

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
namespace GeoWebServis
{
    public class Service1 : System.Web.Services.WebService
    {
        public Service1()
        {
            InitializeComponent();
        }
    }
}
```

```
#region Component Designer generated code
private IContainer components = null;
private void InitializeComponent()
{
}
protected override void Dispose( bool disposing )
{
    if(disposing && components != null)
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}
#endregion
//      [WebMethod]
//      public string HelloWorld()
//      {
//          return "Hello World";
//      }
//  }
}
```

İlk olarak, servismizin adını değiştireceğiz. Bunun için öncelikle, Solution Explorer kısmında, Service1.asmx dosyasının ismini, GeoMat.asmx olarak değiştirelim.



Şekil 2. Service isminin değiştirilmesi.

Şimdi ise, daha önce Notepad uygulamamızda yazdığımız kodları buradaki GeoMat.asmx.cs Code-Behind dosyamız içinde aynen yazıyoruz.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
namespace GeoWebServis
{
    [WebService(Namespace="http://ilk/servis/",Description="Geometrik Hesaplamalar Üzerine Metodlar İçerir.
    Ucgen, Dortgen gibi şekillere yönelik alan ve çevre hesaplamaları.",Name="Geometrik Hesaplamalar")]
    public class TemelIsler : System.Web.Services.WebService
    {
        private const double pi=3.14;
        [WebMethod(Description="Daire Alan Hesabı Yapar")]
        public double DaireAlan( double r)
        {
            return (r*r)*pi;
        }
    }
}
```

[WebMethod(Description="Daire Çevre Hesabı Yapar.")]

```

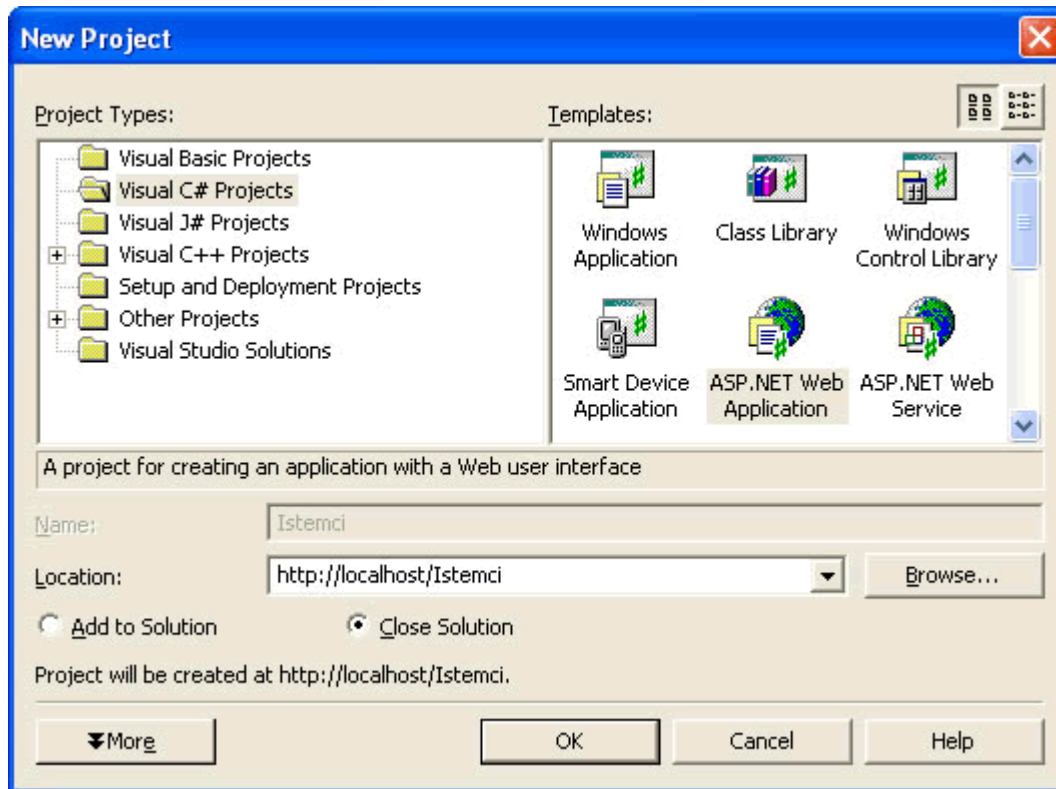
public double DaireCevre( double r)
{
    return 2*pi*r;
}
}
}

```

Projemizi derlediğimizde, Visual Studio.Net, web servisimizin Code-Behind dosyasını kullanarak, servisimiz için gerekli dll dosyasını otomatik olarak bin klasörü içerisinde oluşturacaktır. Hazır olan web servisini test etmek için, projeyi çalıştırmamız (Run) yeterlidir.

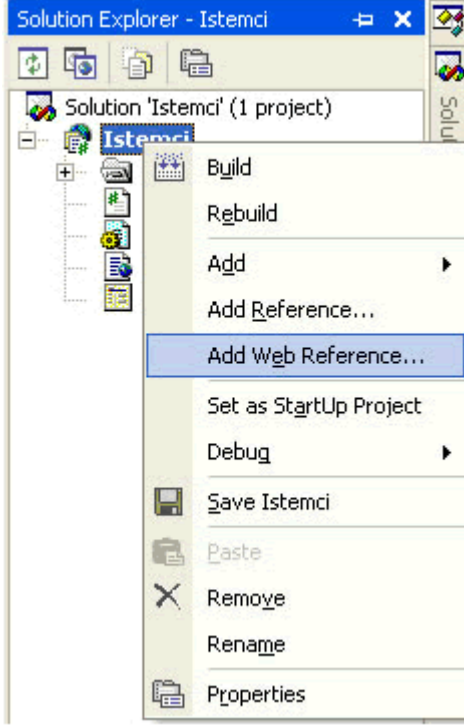
Bir web servisi uygulaması elbette tek başına bir anlam ifade etmez. Web servisleri, ancak istemciler tarafından kullanıldıkları takdirde anlam kazanacaktır. Bir web servisinin, herhangi bir istemci uygulamada kullanılabilmesi yüzeysel olarak bakıldığında çok karmaşık değildir. İstemcinin herşeyden önce, web servisinin yer aldığı adrese (URL) bir referansta bulunması gerekir. Bu referansın geçerli olması halinde, bir sonraki adımda istemci uygulamanın servisin sahip olduğu yetenekleri bilmesi gerekmektedir. Bu mimarinin işleyişini ilerleyen makalelerimizde daha derinlemesine incelemeden önce basit olarak geliştirdiğimiz web servisini bir istemci uygulamada nasıl kullanacağımızı görelim.

İstemci uygulamamızı şimdilik Visual Studio.Net ortamında geliştireceğiz. Mimarının temellerini daha iyi kavradıktan sonra bu işin komut satırı araçları ile nasıl gerçekleştirildiğini daha iyi anlayacağız. Şimdi, Visual Studio.Net ortamında bir uygulama oluşturalım. Bu uygulama bir Windows uygulaması, bir Asp.net uygulaması olabileceği gibi bir Mobil uygulamada olabilir. Şimdi, New Project bölümünden, Asp.Net Web Application tipini seçelim ve uygulamamızı İstemci ismi ile oluşturalım.



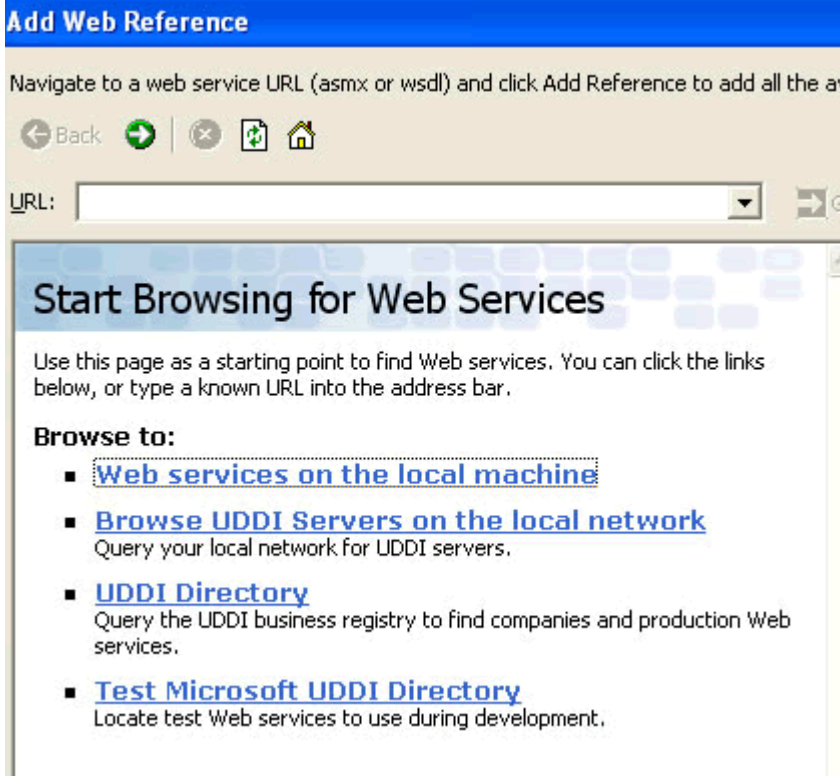
Şekil 3. Yeni bir web uygulamasının açılması.

Sırada bir web servisini uygulamamıza ekleyeceğimiz en önemli kısım var. Bunun için Solution Explorer’ da projemize sağ tıklayıyor ve açılan menüden Add Web Reference öğesini tıklıyoruz.



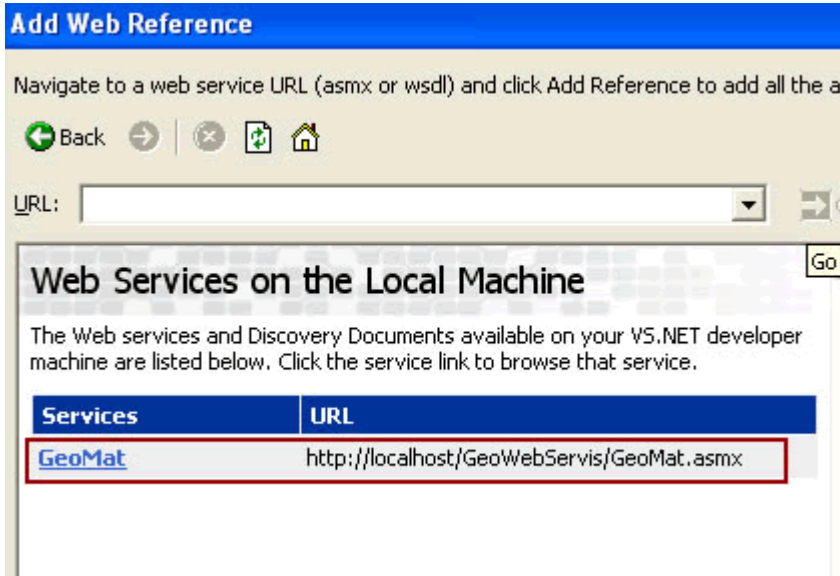
Şekil 4. Web servisi için gerekli referansın eklenmesi.

Bu işlem sonucunda ekrana, web servisini ekleyebilmemiz için bir kaç yöntem sunan **Add Web Reference** dialog penceresi gelecektir.



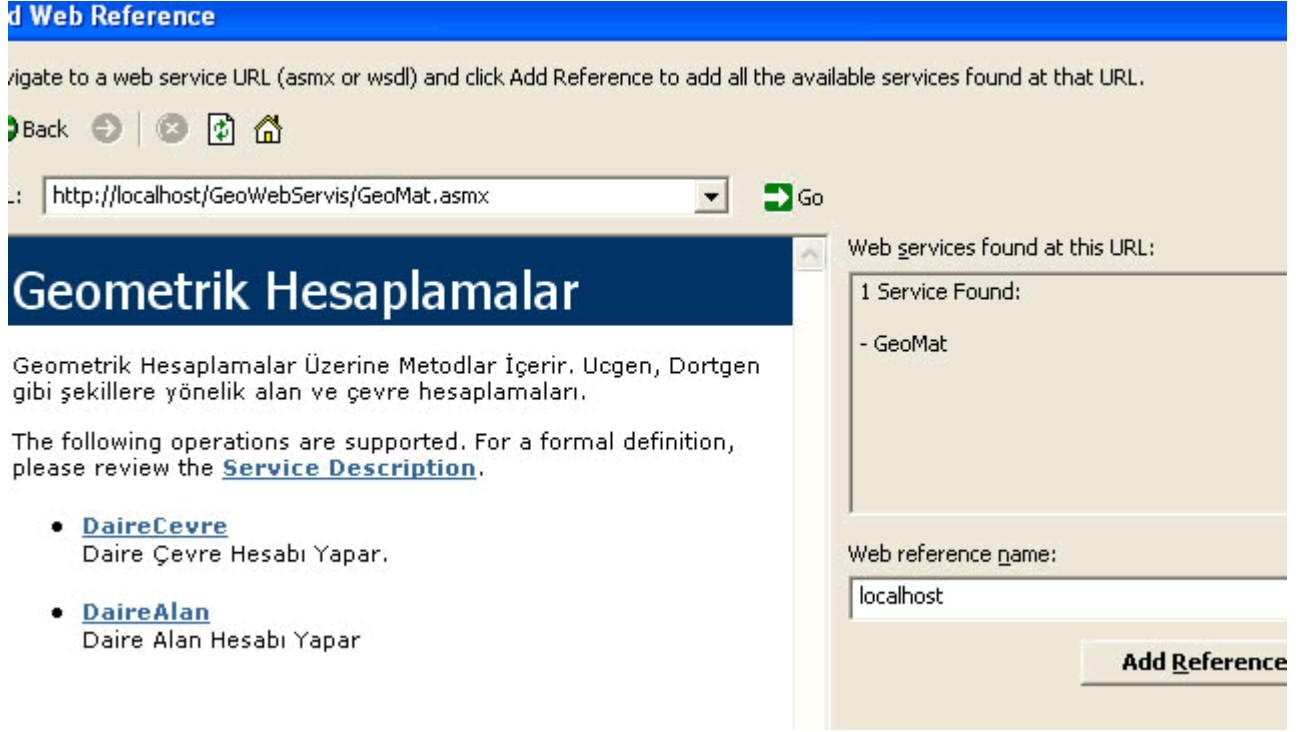
Şekil 5. Web servisinin aranması.

Servisimizi, yerel makinede geliştirdiğimiz için, **Web services on the local machine** bağlantısına tıklamamız halinde, localhost' ta yer alan kullanılabilir tüm web servislerinin listesini elde ederiz. Bu linke tıkladığımızda, geliştirmiş olduğumuz Web Servisini ve bu servise ulaşabileceğimiz adresi elde ederiz.



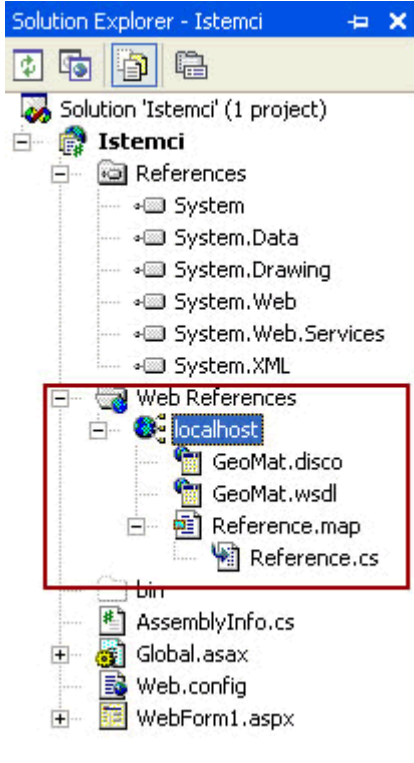
Şekil 6. Yerel sunucuda bulunan web servisi.

GeoMat isimli bağlantıya tıkladığımızda, web servisine ait bilgilerin yer aldığı ve web servisini test edebilmemizi sağlayan pencere ile karşılaşırız. Hatırlatmak gerekirse, burada yer alan servise ve metotlara ait temel bilgiler WebService ve WebMethod nitelikleri sayesinde gerçekleştirilmiştir.



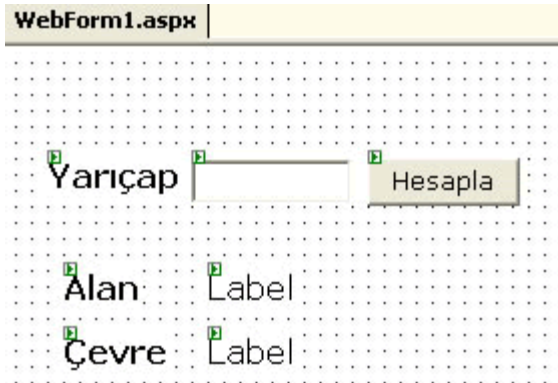
Şekil 7. Servisimizin sağladıklarına bakılması.

Artık web servisimizi bulduğumuza ve ne tür işlemler gerçekleştirebildiğini öğrendiğimize, dolayısıyla servisi keşfettiğimize (Discovery) göre, bu servisi uygulamamıza **Add Reference** butonuna tıklayarak ekleyebiliriz. Bu işlemin ardından, Visual Studio.Net uygulamaya bazı yeni dosyalar ekleyecektir.



Şekil 8. Servisimiz ile birlikte oluşturulan ek dosyalar.

Tüm bu dosyalar Web References sekmesinin altında yer almaktadır. Bu dosyaların ne işe yaradığını ve ne amaçla oluşturulduğunu incelemeden önce, projemize eklediğimiz web servisini uygulama içerisinde nasıl kullanacağımızı görelim. Bu amaçla basit Web Form’umuzu aşağıdaki gibi tasarlayalım.



Şekil 9. Web sayfamız.

Şimdi, Hesapla başlıklı butona basıldığında meydana gelecek işlemleri kodlayacağız. Burada, web servisimizdeki Alan ve Çevre isimli metodları çağırarak ve bu metodlara parametre olarak TextBox metin kutusu kontrolüne girilen değeri göndereceğiz. Daha sonra Web Servisimiz bu metin kutusundaki değeri ilgili metodlarda çalıştıracak ve işlemlerin sonucu olarak metodlardan dönen değerleri, web sayfamıza gönderecek. Bunu gerçekleştirebilmek için tek yapmamız gereken aşağıdaki kodları yazmak.

```
private void btnHesapla_Click(object sender, System.EventArgs e)
{
    double r=Convert.ToDouble(txtYaricap.Text);
    localhost.GeometrikHesaplamalar gh=new Istemci.localhost.GeometrikHesaplamalar();
    lblAlan.Text=gh.DaireAlan(r).ToString();
    lblCevre.Text=gh.DaireCevre(r).ToString();
}
```

Burada belkide en önemli kısım, Web Servisimizde yer alan GemotrikHesaplamalar isimli sınıfa ait bir nesne örneğinin aşağıdaki kod satırı ile oluşturulmasıdır.

```
localhost.GeometrikHesaplamalar gh=new Istemci.localhost.GeometrikHesaplamalar();
```

Web servisimizdeki metodlara işte bu nesne vasıtasıyla erişmekteyiz.

```
lblAlan.Text=gh.DaireAlan(r).ToString();
lblCevre.Text=gh.DaireCevre(r).ToString();
```

Burada kullanılan web servisinin, global seviyedeki bir ağda yer aldığını düşünürsek, servis içindeki sınıfa ait nesnenin istemci makinede nasıl örneklenebildiğini anlamak önemlidir. Nitekim sınıfa ait nesne örneğinin oluşturulabilmesi, istemci makinede de, bu sınıfın olmasını gerektirir. Bu Remoting teknolojisinden gelen bir özelliktir. Bir Remoting uygulamasında, uzak hizmetleri sağlayan nesne modelleri (dll dosyaları), bu hizmetleri kullanmak isteyen makinelere kopyalanır ve uygulamalara referans olarak belirtilir. Oysaki web servislerinde durum farklıdır. Web servisimize ait sınıfı, istemci uygulamaya kopyalamadık. Peki nasıl oldu da bu servis sınıfı, istemci uygulamada kullanılabilir. İşte WSDL (Web Services Description Language) bu noktada devreye girerek bize yardımcı olmaktadır. Mimarının daha derinlerine girerek bu konuyu net bir şekilde kavramadan önce uygulamamızın nasıl çalıştığına bakalım. Bunun için projeyi Run edelim ve metin kutusuna bir yarıçap değeri girelim. Sonuç aşağıdaki gibi olacaktır.

Yarıçap

Alan 314

Çevre 62,8

Şekil 10. Web servisinin çalıştırılması sonucu.

Artık web servisimizi kullanan bir istemci uygulamamız var. Peki gerçekte, kamera arkasında olanlar neler. Nasıl oluyorda, web servisini kullanmak istediğimizde, (bu servis dünyanın başka ucundaki bir makinede olsa bile) bu servise ait bir nesne örneğini istemci uygulamada oluşturabiliyoruz? Hatta bu nesne nasıl oluyorda, web servisindeki metodları biliyor ve onları parametreleri ile birlikte çağırabiliyor? İşte tüm bu soruların cevabı için öncelikle mimariye daha derinlemesine bakmamız gerekiyor. Web servislerinin mimarisini ilerleyen makalelerimizde incelemeye çalışacağız. Tekrar görüşünceye dek hepinize mutlu günler.

Xml Web Servisleri - 3 (Mimarinin Temelleri - SOAP)

Değerli Okurlarım, Merhabalar.

Bu makalemizde, Xml Web Servislerinin mimarisine daha yakında bakmaya çalışacak ve SOAP (Simple Object Access Protocol) 'ı kısaca tanımaya çalışacağız.

Bir web servisinin, istemci uygulamalar tarafından nasıl kullanılabildiğini anlamak, web servislerinin mimarisini iyi bilmekle mümkündür. Mimariyi kolay bir şekilde anlayabilmek için, daha önceki makalemizde geliştirdiğimiz web servisi ve istemci uygulamayı göz önüne alacağız. Herşeyden önce geliştirdiğimiz web servisi local olarak test edilebilen ve tarayıcı üzerinde çalışabilen bir asmx dosyasından ve buna bağlı Code-Behind dosyasından oluşmaktadır. Web servisini test etmek için, web servisinin bulunduğu adresteki asmx uzantılı dosyayı, tarayıcı penceresinden çalıştırmak yeterlidir. Bunun sonucunda, tarayıcı penceresinde bu web servisi hakkındaki bilgilere ulaşabilir ve içerdiği metodları görebiliriz.

Ancak burada yer alan Service Description bağlantısı bize başka bir olanak daha sağlamaktadır. Bu bağlantı yardımıyla web servisimizin tüm içeriğini anlatan bir WSDL dökümanına erişebiliriz. WSDL dökümanının en önemli yanı, XML tabanlı bir içeriğe sahip olmasıdır. Diğer yandan bu döküman, web servisinde kullanılabilecek tüm metodlara, parametrelere ve dönüş değerlerine ilişkin bilgileri içermektedir.

Peki bu WSDL dökümanı ne için oluşturulur? İşte bu noktada istemci uygulamaya bir göz atmakta fayda vardır. İstemci uygulamanın web servisini kullanabilmesi için, ilk önce web servisinin bulunduğu adrese başvurması gerekir. Bu başvurunun ardından web servisine ait referansı istemci uygulamaya eklediğimizde, bir takım yeni dosyalarında uygulamaya eklendiğini görürüz. Bu dosyalardan belkide en önemli olanı Reference.cs isimli dosyadır. Geliştirdiğimiz uygulama ele alındığında Reference.cs dosyasının içeriği aşağıdaki gibi olacaktır.

```
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Web.Services;
```

```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="Geometrik HesaplamalarSoap", Namespace="http://ilk/servis/")]
public class GeometrikHesaplamalar : System.Web.Services.Protocols.SoapHttpClientProtocol
{
    public GeometrikHesaplamalar()
    {
        this.Url = "http://localhost/GeoWebServis/GeoMat.asmx";
    }
}
```

```

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://ilk/servis/DaireAlan",
RequestNamespace="http://ilk/servis/", ResponseNamespace="http://ilk/servis/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
public System.Double DaireAlan(System.Double r)
{
    object[] results = this.Invoke("DaireAlan", new object[] {r});
    return ((System.Double)(results[0]));
}

public System.IAsyncResult BeginDaireAlan(System.Double r, System.AsyncCallback callback, object asyncState)
{
    return this.BeginInvoke("DaireAlan", new object[] {r}, callback, asyncState);
}

public System.Double EndDaireAlan(System.IAsyncResult asyncResult)
{
    object[] results = this.EndInvoke(asyncResult);
    return ((System.Double)(results[0]));
}

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://ilk/servis/DaireCevre",
RequestNamespace="http://ilk/servis/", ResponseNamespace="http://ilk/servis/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
public System.Double DaireCevre(System.Double r)
{
    object[] results = this.Invoke("DaireCevre", new object[] {r});
    return ((System.Double)(results[0]));
}

public System.IAsyncResult BeginDaireCevre(System.Double r, System.AsyncCallback callback, object asyncState)
{
    return this.BeginInvoke("DaireCevre", new object[] {r}, callback, asyncState);
}

public System.Double EndDaireCevre(System.IAsyncResult asyncResult)
{
    object[] results = this.EndInvoke(asyncResult);

```

```
        return ((System.Double)(results[0]));  
    }  
}
```

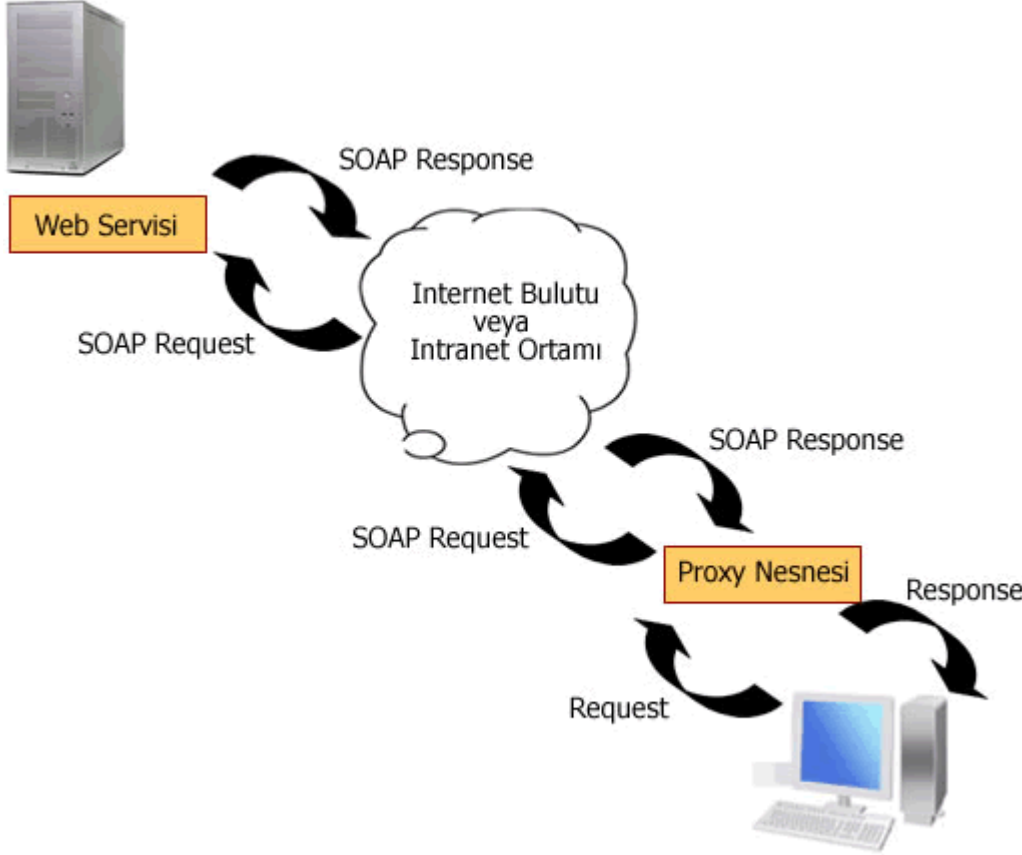
Visual Studio.NET tarafından otomatik olarak oluşturulan bu dosyada dikkat çekici noktalar vardır. Herşeyden önce, karşımızda, web servisimizin bir görüntüsü yer almaktadır. Web servisimizde yazdığımız metodlar kullandıkları parametreler ve daha başka bilgiler. Ancak önemli olan bu dosya sayesinde, istemci uygulamanın artık web servisine ait bir nesne örneğini oluşturup kullanabilecek olmasıdır. Dolayısıyla, istemci uygulamamıza web servisimize ait referansı eklediğimizde, istemci uygulamada bu servise ait bir nesne yapısı oluşturulabilmiştir. Bu sayede aşağıdaki gibi bir bildirim geçerli hale gelir.

```
localhost.GeometrikHesaplamalar gh=new Istemci.localhost.GeometrikHesaplamalar();
```

Dahası, bu nesne üzerinden, web servisindeki metodları aynı isimler ile kullanabiliriz.

```
lblAlan.Text=gh.DaireAlan(r).ToString();  
lblCevre.Text=gh.DaireCevre(r).ToString();
```

Olaya daha detaylı bakıldığında, Reference.cs' nin aslında, istemci ve web servisi arasındaki haberleşmeyi sağlayacak bir Proxy nesnesini oluşturmak amacıyla kullanıldığını söyleyebiliriz. Dolayısıyla, istemci uygulama bu servisi kullanmak istediğinde, yani bu servis üzerinden bir metodu çağırmak istediğinde, bu talebi proxy nesnesinden ister. Proxy nesnesi ise bu talebi , web servisine iletir. Web servisi gelen talebi değerlendirir ve ürettiği cevabı yine istemci uygulamadaki proxy nesnesine gönderir. Proxy nesneside sonuçları, uygulama ortamına iletir.

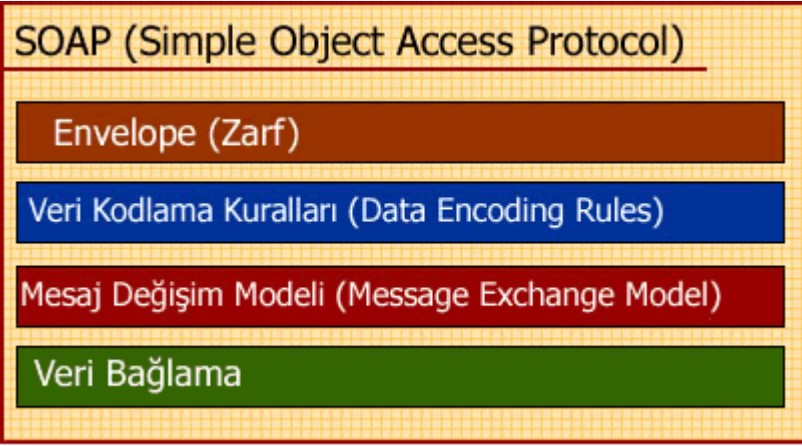


Şekil 1. Proxy Nesnesini ve SOAP' ın Xml Web Servisi Mimarisindeki yeri.

Diğer bir sorun, proxy nesnesine gelen taleplerin, web servisine giderken hiç bir engel ile karşılaşmadan nasıl hareket edeceği ve geri geleceğidir. İstemci uygulama, proxy nesnesinde normal olarak talepte bulunur ve cevapları alır. Buradaki ilişki normal olarak bir nesne.metod ilişkisidir. Ancak proxy nesnesi bu mesajları, esnek, kolay okunabilir, herhangi bir engele takılmayacak bir hale getirmek durumundadır. Bu iş için XML tabanlı bir bilgi akışı biçilmiş kaftandır. Yinede hareket edecek mesajların uygun bir formasyonda taşınmaları ve web servisinin anlayabileceği bir dilde ifade edilebilmeleri daha doğrudur. Bu noktada **SOAP (Simple Object Access Protocol – Basit Nesne Erişim Antlaşması)** devreye girer.

Web servislerinin kullanılmasında, web servisleri ve istemciler arasındaki haberleşmenin belirli standartlar çerçevesinde geliştirilmesi çok önemlidir. SOAP (Simple Access Object Protocol) işte bu noktada devreye giren ve web servisi-istemci sisteminin en önemli kısmını oluşturan bir yapı taşıdır. SOAP, web servisleri ve istemciler arasında gidip gelecek mesajların, XML tabanlı olarak belirlendiği standartlara uygun formatta taşınmasını sağlayan bir protokoldür. Çoğunlukla HTTP üzerinde çalışan SOAP, FTP ve SMTP gidi diğer iletişim protokolleri üzerinden de kullanılabilir.

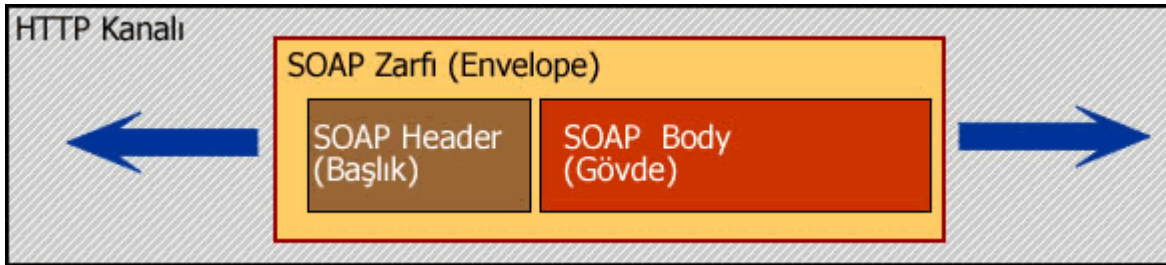
SOAP protokolü, web servisleri ile istemciler arasında gerçekleştirilen veri alışverişinde, karşılıklı olarak akıcak mesajların nasıl ve ne şekilde paketleneceğini yada başka bir deyişle bilgilerin nasıl kapsülleneceğini belirtir. SOAP, özünde XML tabanlı mesajların oluşturulmasını belirtir. Bu nedenle SOAP protokolünü uygulayan mesajlar (ki bunlar SOAP Mesajı olarak adlandırılır), herhangi bir ağ ortamında hiç bir sorunla karşılaşmadan uzak makineler arasında iletilirler. SOAP protokolü, 4 temel üzerine inşa edilmiştir.



Şekil 2. SOAP Protokolünün Temelleri.

SOAP protokolünün dayandığı bu temel kurallar içerisinde en önemlisi Envelope kısmıdır. Bir SOAP mesajı mutlaka bir zarf olarak teşkil edilmeli ve Zarf kurallarına göre tasarlanmalıdır. Diğer temellerin uygulanması zorunlu değildir. Veri kodlama kuralları, özellikle serileştirilen nesneler için bir model sunar. Başka bir deyişle, tanımlanmış veri tiplerinin uygulama için kullanılma kurallarına karar verir. Bu katmandaki kuralların uygulanması opsiyoneldir. Mesaj değişim modeli, web servisi ile istemciler arasında değiş tokuş edilen mesajlar için bir istek/cevap deseni tanımlar. SOAP, Remote Procedure Call tip mekanizmasını esas alan bir veri değiş tokuş desenini kullansada bu bir zorunluluk değildir. Mesaj değişim modelide opsiyoneldir. Veri bağlama kuralları ile, SOAP'ın iletişim protokollerinin birbirlerine nasıl bağlanacağına dair tanımlamalar içerir. Veri bağlama kurallarıda opsiyoneldir.

Kısaca SOAP protokolü, mutlaka iletişimsel mesajların SOAP Zarf kurallarına uygun bir biçimde tasarlanmasını zorunlu kılar. SOAP mesajlarının zarf kurallarına uygun bir biçimde tasarlanması için aşağıdaki şekilde belirtilen format kullanılır. Burada bir SOAP zarfının temel yapısı belirtilmiştir.



Şekil 3. SOAP Zarfının Temel Yapısı.

SOAP Zarfları bir SOAP mesajının başlık ve gövde olmak üzere iki ana kısımdan oluşması gerektiğini belirtir. En önemli kısım gövdedir. Burada, yapılan metod çağırımlarına ilişkin bilgiler ile çağrı sonucu istemcilere gönderilecek cevaplara ait xml tabanlı bilgiler yer alır.

SOAP mesajlarını anlamanın en iyi yolu, onları gerçek bir uygulamada takip etmektir. Bu amaçla, geliştirmiş olduğumuz Web servisini ve istemci uygulamamızı kullanacağız. İstemci ve web servisi arasında hareket eden SOAP mesajlarını takip edebilmek amacıyla Microsoft firmasının sunduğu **SOAP Toolkit** aracını kullanabiliriz. Bu aracı bugün itibarıyla <http://download.microsoft.com/download/2/e/0/2e068a11-9ef7-45f5-820f-89573d7c4939/soapsdk.exe> adresinden temin edebilirsiniz. SOAP mesajlarını takip edebilmek amacıyla istemci uygulamamızda ufak bir değişiklik yapmamız gerekiyor. Bunun için

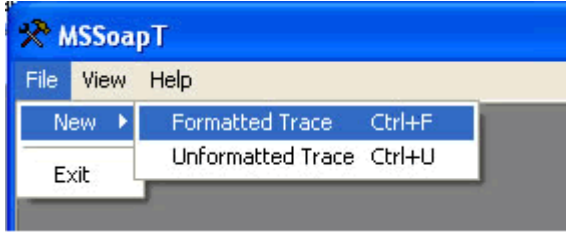
istemci uygulamda, web servisine ait proxy nesnesinin sınıf dosyası içerisinde (reference.cs) yer alan,

```
this.Url = "http://localhost/GeoWebServis/GeoMat.asmx";
```

satırını

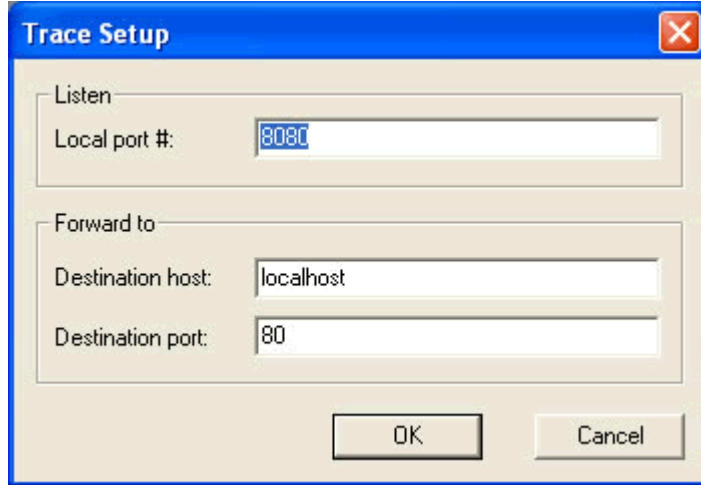
```
this.Url = "http://localhost:8080/GeoWebServis/GeoMat.asmx";
```

şeklinde değiştirmeliyiz. Şimdi SOAP Toolkit 3.0 ile yüklenen **Trace Utility** aracını çalıştıralım. Ardından Trace Utility aracında, **New** menüsünden, **Formatted Trace**’ i seğıelim.



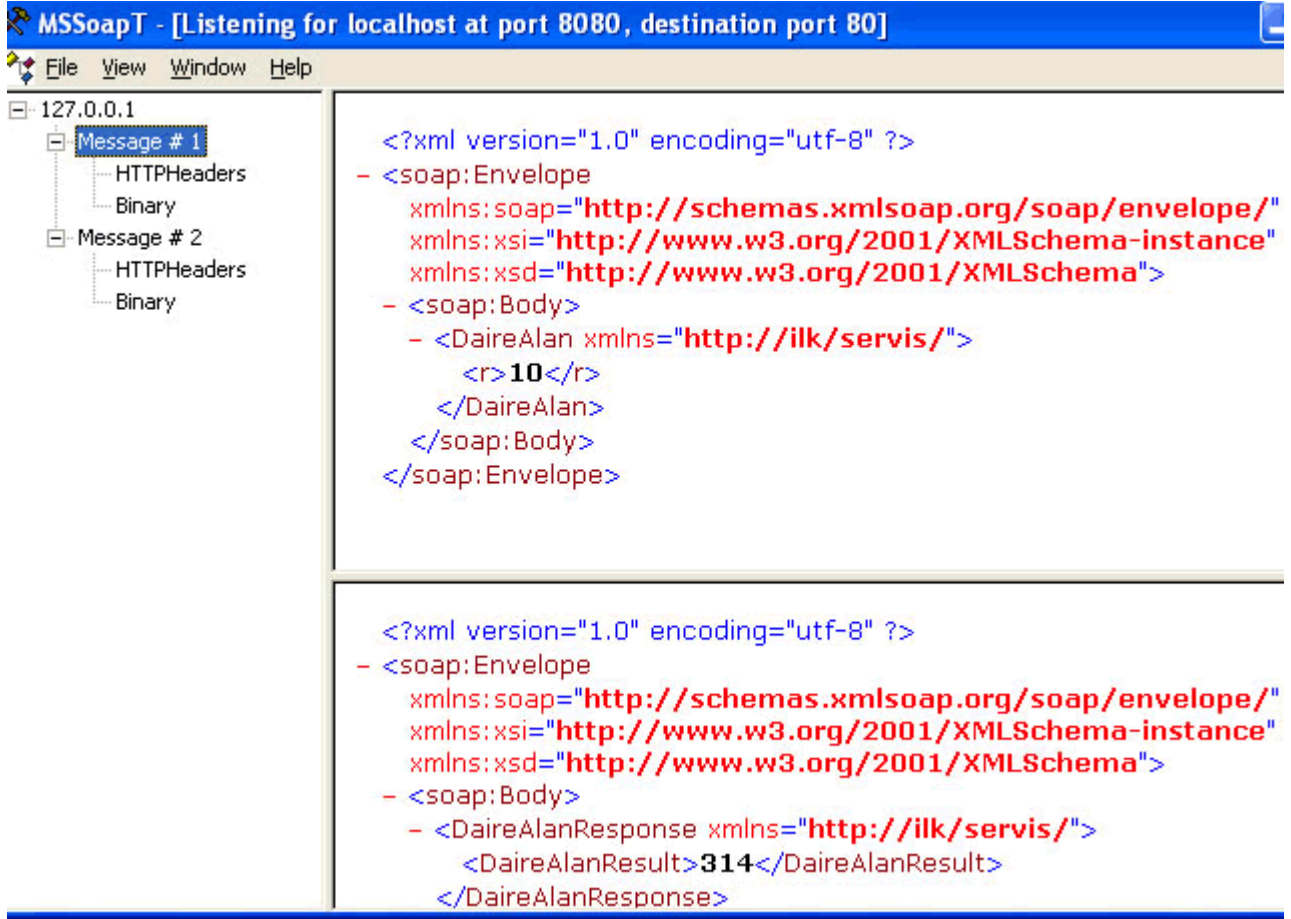
Şekil 4. Yeni bir Trace açıyoruz.

Karşımıza çıkacak olan aşağıdaki pencereyi bu hali ile bırakalım.



Şekil 5. Trace Setup.

Ardından istemci uygulamamızı çalıştıralım ve web servisini kullanalım. Tekrardan Trace Utility pencremize dönelim. Aşağıdaki ekran görüntüsü elde ederiz. Görüldüğü gibi, trace utility aracı, web servisi ile istemci uygulamamız arasındaki SOAP mesajlarını yakalamıştır. Burada iki mesaj sekmesi görünmektedir. Bunların herbiri web servisi içindeki bir metoda karşılık gelmektedir.



Şekil 6. SOAP Mesajları.

Örneğin Message #1 sekmesini inceleyelim. Üsteki XML bilgisi, istemcinin web servisine gönderdiği talebi (Request) göstermektedir. Altteki mesaj ise, web servisinden istemciye gelen cevabı (Response) gösterir. İstemcinin talebini incelediğimizde aşağıdaki XML dökümanını elde ederiz.

```
<?xml version="1.0" encoding="utf-8" ?>
- <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- <soap:Body>
- <DaireAlan xmlns="http://ilk/servis/">
  <r>10</r>
</DaireAlan>
</soap:Body>
</soap:Envelope>
```

Burada istemciden web servisine giden SOAP mesajı görülmektedir. Görüldüğü gibi mesaj <soap:Envelope tag'ı ile başlamaktadır. Soap mesajımızın gövdesi ise <soap:Body> tagı ile başlar. Burada istemci tarafından çağırılan metodun ismi ve bu metoda gönderilen parametre değeri yer almaktadır. Ayrıca metodun hangi isim alanı içinde yer aldığıda belirtilmiştir. Bu isim alanının web

servisin içerisindeki WebService niteliği ile belirttiğimiz isim alanı olduğuna dikkat ediniz. Soap mesajımızın başlık kısmı ise, HTTP Header sekmesinde görülebilir. Buradaki XML satırları içerisinde en önemlisi <soapaction> tagının olduğu satırdır.

```
<soapaction>"http://ilk/servis/DaireAlan"</soapaction>
```

Soapaction tagı, kullanılan web servisinde belirtmiş olduğumuz xml isim alanının sonuna web servisinde yer alan metodun ismi eklenerek oluşturulmuştur. Burada, Soap mesajının web servisi tarafından çözümlenmesi sırasında, karşılık gelecek isim alanının (WebService sekmesinde belirttiğimiz) ve bu isim alanı içinden çağırılan metodun (WebMethod niteliği ile belirttiğimiz) tanımlamaları yer alır. Böylece web servisi gelen soap mesajına karşılık gelen metodu bulup çalıştırabilir. Gelelim web servisten istemciye dönen SOAP Mesajına.

```
<?xml version="1.0" encoding="utf-8" ?>
- <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- <soap:Body>
- <DaireAlanResponse xmlns="http://ilk/servis/">
  <DaireAlanResult>314</DaireAlanResult>
</DaireAlanResponse>
</soap:Body>
</soap:Envelope>
```

Görüldüğü gibi, web servisten istemciye dönen Soap mesajında en dikkat çekici nokta Response ve Result anahtar kelimeleridir. İstemci tarafından çözümlenecek olan bu Soap mesajı, istemci tarafından metodun çağırıldığı satıra dönecek geri dönüş değerlerini içerir. Yine buradaki gövdede yer alan metod tanımında kullanılan isim alanı ile (xmlns="http://ilk/servis/"), istemciye ait proxy nesnesini oluşturan sınıf içerisindeki WebService niteliğinde kullanılan isim alanının aynı olduğuna dikkat edelim.

```
[System.Web.Services.WebServiceBindingAttribute(Name="Geometrik HesaplamalarSoap", Namespace="http://ilk/servis/")]
```

Burada görüldüğü gibi SOAP mesajlarımız tek parametre alan metodlar ve tek sonuç içeren geri dönüş değerlerini istemci ve web servisi arasında belirli bir standart dahilinde taşımaktadır. Diğer yandan, özellikle değişken sayıda parametre alan ve geriye dizi veya veri kümesi döndürebilen SOAP mesajlarda mümkündür. Bu durumu analiz edebilmek amacıyla web servisimize parametre olarak bir dizi alan ve geriye bu dizinin işlenmiş halini döndürecek bir metod ilave edelim.

```
[WebMethod(Description="Daire Çevre Hesabını Dizi Elemanlarına Uygular.")]
```

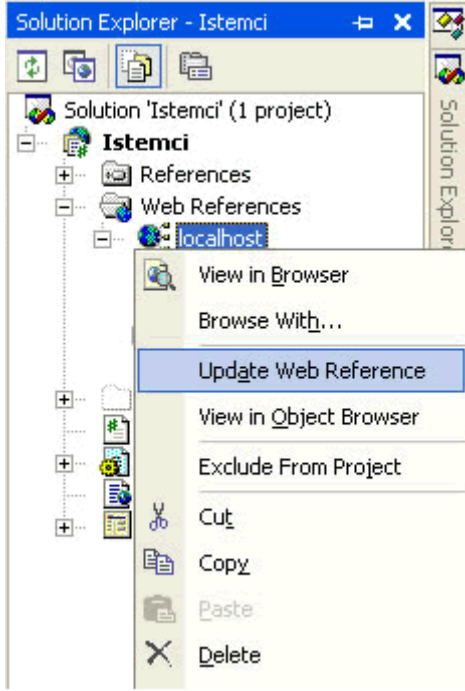
```
public double[] DaireCevreDizi( double[] r)
{
    int eleman_Sayisi=r.Length;
    double[] dizi=new double[eleman_Sayisi];
    for(int i=0;i<eleman_Sayisi;i++)
    {
        dizi[i]=r[i]*pi*2;
    }
}
```

```

return dizi;
}

```

Bu web metodumuz, istemciden double türünden bir dizi alacak ve bu dizideki elemanlara çevre hesaplaması işlemlerini uygulayacak. Sonuçlar ise yine bir dizi şeklinde istemci bilgisayara geri döndürülecek. İstemci uygulamamızda bu yeni metodu test edebilmemiz için, proxy nesnesinin, web servisinin yeni versiyonuna uygun bir şekilde güncellenmesi gerekir. Bunun için, istemci uygulamada Solution Explorer penceresinde, Web References sekeminde localhost ögesine sağ tıklayarak açılan menüden, **Update Web Reference**' i seçmemiz yeterli olacaktır.



Şekil 7. Xml Web Servisini Güncellemek.

Bu durumda, istemci uygulamamız web servisimiz için oluşturduğu proxy nesnesini yeniliklere göre güncelleyecektir. Aksi takdirde web servisimize eklediğimiz metodu kullanamayız. Bu işlemlerin ardından, istemci uygulamamıza aşağıdaki kod satırlarını ilave edelim.

```

double[] dizi=new double[3];
dizi[0]=1;
dizi[1]=2;
dizi[2]=3;

double[] diziSonuc=new double[3];
diziSonuc=gh.DaireCevreDizi(dizi);
foreach(double eleman in diziSonuc)
{

```

```

        lblDizi.Text+=eleman.ToString()+" ";
    }

```

Burada yaptığımız, 3 elemanlı double türünden bir dizi tanımlamak ve bu diziye atadığımız elemanları, web servisimizdeki metodumuza parametre olarak gönderip sonuçları bir label kontrolünde yazdırmak. Bizim için önemli olan kodların yazılış tarzından çok, web servisine bir dizinin gönderilmesi, orada işlenmesi ve sonuçların geri gelmesi sırasında, SOAP mesajlarının ne şekilde oluşacağıdır.

Şimdi Trace Utility 'den SOAP mesajlarını yenide inceleyelim. Yerel makinede çalıştığımız ve proxy nesnemizide güncellediğimiz için, proxy sınıfındaki localhost tanımlamasını tekrardan localhost:8080 olarak değiştirmemiz, SOAP mesajlarını Trace Utility aracılığıyla yakalayabilmemiz açısından gerekli olabilir. SOAP Toolkit Trace Utility aracıyla, SOAP mesajlarına baktığımızda, istemciden web servisine DaireCevreDizi metodu için giden XML tabanlı bilgilerin aşağıdaki şekilde oluşturulduğunu görürüz.

```

<?xml version="1.0" encoding="utf-8" ?>
- <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- <soap:Body>
- <DaireCevreDizi xmlns="http://ilk/servis/">
- <r>
    <double>1</double>
    <double>2</double>
    <double>3</double>
  </r>
</DaireCevreDizi>
</soap:Body>
</soap:Envelope>

```

Görüldüğü gibi web metoda gönderdiğimiz dizi elemanları double XML veri tipinden elemanlar olarak SOAP mesajının gövdesine katılmıştır. Aynı şekilde web servisinden geri dönen mesaja baktığımızda aşağıdaki sonucu elde ederiz.

```

<?xml version="1.0" encoding="utf-8" ?>
- <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- <soap:Body>
- <DaireCevreDiziResponse xmlns="http://ilk/servis/">
- <DaireCevreDiziResult>
    <double>6.28</double>
    <double>12.56</double>
    <double>18.84</double>
  </DaireCevreDiziResult>
</DaireCevreDiziResponse>

```

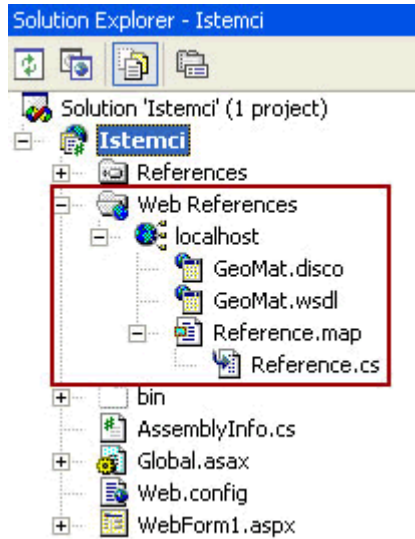
```
</soap:Body>
</soap:Envelope>
```

Sonuçlar yine double XML veri tipi elemanı olarak geri döndürülmüştür. Şu ana kadar yaptığımız örneklerde, istemci uygulamanın web servisindeki sınıf ve metodları bir proxy nesnesi yardımıyla nasıl kullandığına şahit olduk. Peki istemci bir uygulama, kendisine web referansı olarak eklediği web servisine dair bir proxy nesnesini nasıl oluşturabildi? Bu sorunun cevabını Web Servislerinin SOAP' tan sonra olmasa olmaz materyali olan WSDL verebilir. WSDL konusunu bir sonraki makalemizde incelemeye çalışacağız. Tekrarda görüşünceye dek hepinize mutlu günler dilerim.

Xml Web Servisleri - 4 (Mimarının Temelleri - WSDL)

Değerli Okurlarım, Merhabalar.

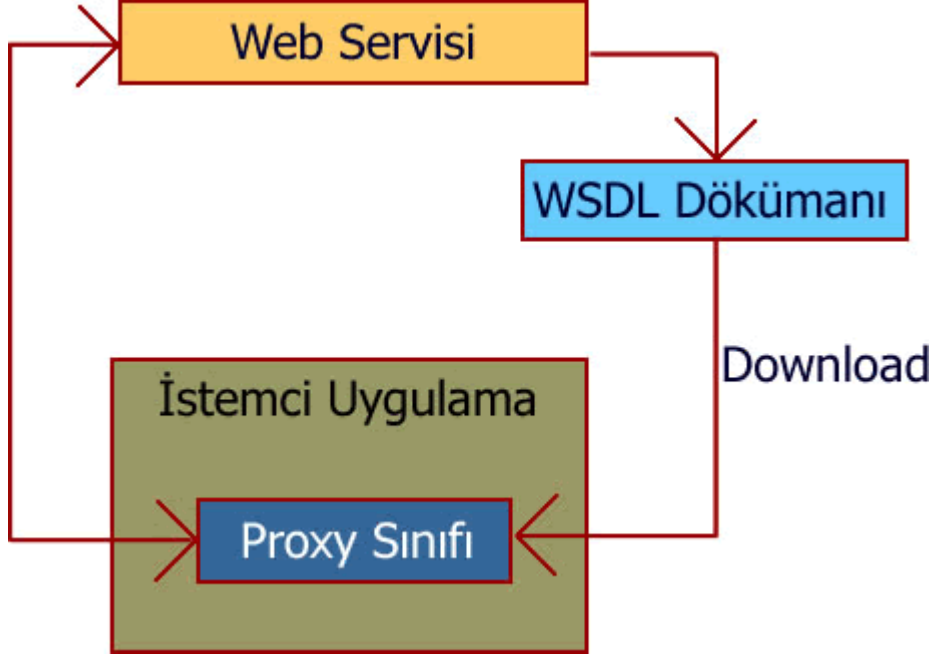
İstemciler, web servisleri ile aralarındaki iletişimi, çalıştıkları makinede oluşturulan proxy nesneleri yardımıyla gerçekleştirir. Bu, istemci uygulamanın, web servisine ait üyelerin farkında olmasını gerektiren bir durumdur. Nitekim proxy nesnesini oluşturan sınıf, web servisindeki public arayüze göre tasarlanır. Dolayısıyla, istemci uygulamanın kullandığı web servisine ait bilgileri bir şekilde temin etmesi gerekmektedir. Visual Studio.Net ortamında geliştirdiğimiz istemci uygulamada, projeye web servisinin referans olarak eklenmesi sonucu oluşturulan bazı dosyalar olduğundan bahsetmiştik. Disco uzantılı bir dosya, wsdl uzantılı bir dosya ve proxy sınıfımıza ait cs uzantılı dosya.



Şekil 1. İstemci uygulamada oluşturulan dosyalar.

Proxy sınıfı ile wsdl dosyası arasında sıkı bir ilişki vardır. Visual Studio.Net ortamını göz önüne aldığımızda, istemci uygulamaya Add Web Reference ile var olan bir web servisinin eklenmesi durumunda ilk olarak .net, bu servisten wsdl uzantılı bir döküman talep edecektir. WSDL (Web Service Description Language - Web Servisleri Tanımlama Dili) ile oluşturulan bu döküman XML tabanlı olup, web servisinin herkese açık olan üyeleri, başka bir deyişle arayüzü hakkında tanımlamalara sahiptir. Talep edilen wsdl belgesi istemci uygulamaya indirilir. İndirilen döküman yardımıyla, istemcilerin web servisleri ile haberleşmelerini kolaylaştıracak proxy sınıfı oluşturulur.

WSDL dökümanları XML tabanlı olmaları nedeni ile, her türlü platforma indirilebilirler. XML tabanlı olup web servisinin yapısı hakkında tanımlamalara sahip olduklarından, indirme süreleri çok uzun değildir. Bir wsdl dökümanı istemci uygulamaya, web servisi ilk kez referans edildiğinde indirilir. Elbetteki web servisinde yapılacak güncellemelerin, istemci bilgisayar üzerinde gerçekleştirilmesi durumunda bu wsdl dökümanı tekrardan istemci bilgisayara indirilecektir.



Şekil 2. WSDL Dökümanının Web Servislerinin Kullanımındaki Yeri.

Geliştirdiğimiz istemci uygulamaya ait wsdl dökümanı aşağıdaki gibidir.

```

<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://ilk/servis/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://ilk/servis/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
<s:schema elementFormDefault="qualified" targetNamespace="http://ilk/servis/">
<s:element name="DaireAlan">
<s:complexType>
<s:sequence>
<s:element minOccurs="1" maxOccurs="1" name="r" type="s:double" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="DaireAlanResponse">

```

```

<s:complexType>
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="DaireAlanResult" type="s:double" />
  </s:sequence>
</s:complexType>
</s:element>
<s:element name="DaireCevre">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="r" type="s:double" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="DaireCevreResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="DaireCevreResult" type="s:double" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="DaireCevreDizi">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="r" type="s0:ArrayOfDouble" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="ArrayOfDouble">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="double" type="s:double" />
  </s:sequence>
</s:complexType>
<s:element name="DaireCevreDiziResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="DaireCevreDiziResult" type="s0:ArrayOfDouble" />
    </s:sequence>
  </s:complexType>
</s:element>

```

```

</s:schema>
</types>
<message name="DaireAlanSoapIn">
  <part name="parameters" element="s0:DaireAlan" />
</message>
<message name="DaireAlanSoapOut">
  <part name="parameters" element="s0:DaireAlanResponse" />
</message>
<message name="DaireCevreSoapIn">
  <part name="parameters" element="s0:DaireCevre" />
</message>
<message name="DaireCevreSoapOut">
  <part name="parameters" element="s0:DaireCevreResponse" />
</message>
<message name="DaireCevreDiziSoapIn">
  <part name="parameters" element="s0:DaireCevreDizi" />
</message>
<message name="DaireCevreDiziSoapOut">
  <part name="parameters" element="s0:DaireCevreDiziResponse" />
</message>
<portType name="Geometrik_x0020_HesaplamalarSoap">
  <operation name="DaireAlan">
    <documentation>Daire Alan Hesabi Yapar</documentation>
    <input message="s0:DaireAlanSoapIn" />
    <output message="s0:DaireAlanSoapOut" />
  </operation>
  <operation name="DaireCevre">
    <documentation>Daire Çevre Hesabi Yapar.</documentation>
    <input message="s0:DaireCevreSoapIn" />
    <output message="s0:DaireCevreSoapOut" />
  </operation>
  <operation name="DaireCevreDizi">
    <documentation>Daire Çevre Hesabini Dizi Elamanlarına Uygular.</documentation>
    <input message="s0:DaireCevreDiziSoapIn" />
    <output message="s0:DaireCevreDiziSoapOut" />
  </operation>
</portType>
<binding name="Geometrik_x0020_HesaplamalarSoap" type="s0:Geometrik_x0020_HesaplamalarSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />

```

```

<operation name="DaireAlan">
  <soap:operation soapAction="http://ilk/servis/DaireAlan" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="DaireCevre">
  <soap:operation soapAction="http://ilk/servis/DaireCevre" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
<operation name="DaireCevreDizi">
  <soap:operation soapAction="http://ilk/servis/DaireCevreDizi" style="document" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
</binding>
<service name="Geometrik_x0020_Hesaplamalar">
  <documentation>Geometrik Hesaplamalar Üzerine Metodlar İçerir. Ucgen, Dortgen gibi sekillere yönelik alan ve çevre hesaplamalari.</documentation>
  <port name="Geometrik_x0020_HesaplamalarSoap" binding="s0:Geometrik_x0020_HesaplamalarSoap">
    <soap:address location="http://localhost/GeoWebServis/GeoMat.asmx" />
  </port>
</service>
</definitions>

```

Bu wsdl dökümanı, geliştirdiğimiz servise ait tüm bilgileri içeririr. Hangi metodların kullanılabileceği, bu metodların ne çeşit parametreler aldığı ve geriye ne tür değerleri nasıl döndürdüğü gibi bilgileri sunar. Temel olarak bir WSDL dökümanı 5 ana kısımdan oluşur.

WSDL



Şekil 3. Bir WSDL dökümanının temel kısımları.

Types kısmında, web servisi ile ilişkili SOAP mesajlarında taşınacak parametre ve geri dönüş değerlerine ait şema elemanları tanımlanır. Örneğin,

```
<s:element name="DaireAlan">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="r" type="s:double" />
</s:sequence>
</s:complexType>
</s:element>
```

elamanında, DaireAlan isimli parametrenin double tipinden ve r isminde olduğu belirtilmektedir. Aynı şekilde geri döndürecek değer ise, DaireAlanResponse elemanı olarak tanımlanmıştır.

```
<s:element name="DaireAlanResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="DaireAlanResult" type="s:double" />
</s:sequence>
</s:complexType>
</s:element>
```

Message kısmında, web servisinin kabul edeceği ve geri döndüreceği mesajlara ait özet bilgiler yer alır.

```
+ <message name="DaireAlanSoapIn">
  <part name="parameters" element="s0:DaireAlan" />
</message>
+ <message name="DaireAlanSoapOut">
  <part name="parameters" element="s0:DaireAlanResponse" />
</message>
```

```

+ <message name="DaireCevreSoapIn">
  <part name="parameters" element="s0:DaireCevre" />
</message>
+ <message name="DaireCevreSoapOut">
  <part name="parameters" element="s0:DaireCevreResponse" />
</message>
+ <message name="DaireCevreDiziSoapIn">
  <part name="parameters" element="s0:DaireCevreDizi" />
</message>
+ <message name="DaireCevreDiziSoapOut">

```

Görüldüğü gibi, her SOAP mesajı için In ve Out takılı mesaj elemanları mevcuttur. Buradaki mesajlar, belirledikleri metod için input ve output parametrelerini tanımlarlar. Bir başka deyişle, SOAP mesajları içine konan ve SOAP mesajları ile geri dönen parametreler tanımlanmaktadır.

```

-<message name="DaireAlanSoapIn">
  <part name="parameters" element="s0:DaireAlan" />
</message>
- <message name="DaireAlanSoapOut">
  <part name="parameters" element="s0:DaireAlanResponse" />
</message>

```

Örneğin burada, DaireAlan metodu için oluşturulan SOAP Mesajlarının, iki türü tanımlanmıştır. Web servisine gidecek olan SOAP mesajları, DaireAlan elemanı tipinden parametre alacaktır. DaireAlan elemanının types kısmında tanımlanmış olduğuna dikkat edelim. Aynı şekilde, istemciye dönecek olan SOAP mesajıda, DaireAlanResponse elemanı tipinden bir parametre ile gelecektir.

PortType kısmında ise, her bir web servisi metodu için birer operasyon tanımlaması yapılır. Bu sayede, proxy nesnesi üzerindeki bir metod ile web servisi üzerinde kullanılabilir çağrılar gerçekleştirilebilecektir. Başka bir deyişle, web servisi üzerinden gerçekleştirilebilecek operasyonların tanımlamaları yapılmaktadır. Operasyon isimleri, burada görüldüğü gibi web servisindeki metod isimleri ile aynıdır. Buradaki eleman isimleri ile fiziki metodlar binding kısmında eşleştirilecektir.

```

- <portType name="Geometrik_x0020_HesaplamalarSoap">
- <operation name="DaireAlan">
  <documentation>Daire Alan Hesabi Yapar</documentation>
  <input message="s0:DaireAlanSoapIn" />
  <output message="s0:DaireAlanSoapOut" />
</operation>
- <operation name="DaireCevre">
  <documentation>>Daire Çevre Hesabi Yapar.</documentation>
  <input message="s0:DaireCevreSoapIn" />
  <output message="s0:DaireCevreSoapOut" />

```

```

</operation>
- <operation name="DaireCevreDizi">
  <documentation>Daire Cevre Hesabini Dizi Elamanlarına Uygular.</documentation>
  <input message="s0:DaireCevreDiziSoapIn" />
  <output message="s0:DaireCevreDiziSoapOut" />
</operation>
</portType>

```

DaireAlan operasyonunu ele alalım. Bu operasyon gerçekleştiğinde, SOAP mesajlarının nasıl bir formatta olacağı, başka bir deyişle hangi tipte ve isimde elemanlar alacağı, input message ve output message elemanlarının karşılık geldiği message elemanlarında belirtilmiştir. Message elemanları ise, bu mesajların taşıyacağı parametrelerin tipini tanımlar. Bu tipler ise, types kısmında belirtilmiştir.

Binding kısmında, WSDL dökümanındaki her bir operasyon için, bu operasyona web servisinde karşılık gelecek metod tanımlamaları yapılır. Bir başka deyişle her bir operation elemanı için fiziki olarak metod adresleri belirlenir. Bu adresler için kullanılacak operasyonlar belirli olduğu için, bu operasyonlara bağlı mesajlarda, fiziki adreslere bağlanmış olur.

```

<binding name="Geometrik_x0020_HesaplamalarSoap" type="s0:Geometrik_x0020_HesaplamalarSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <operation name="DaireAlan">
  <soap:operation soapAction="http://ilk/servis/DaireAlan" style="document" />
- <input>
  <soap:body use="literal" />
</input>
- <output>
  <soap:body use="literal" />
</output>
</operation>

```

Örneğin, DaireAlan metoduna ilişkin operation elemanı, soapAction anahtar kelimesini takip eden kısımda web servisinin isim alanı ve metodu ile oluşturulmuş url' ye bağlanmıştır. Son olarak service kısmında ise, tanımlanan port' ların gerçekte fiziki olarak hangi adrese bakacağı tanımlanır.

```

- <service name="Geometrik_x0020_Hesaplamalar">
  <documentation>Geometrik Hesaplamalar Üzerine Metodlar İçerir. Ucgen, Dortgen gibi sekillere yönelik alan ve çevre hesaplamalari.</documentation>
- <port name="Geometrik_x0020_HesaplamalarSoap" binding="s0:Geometrik_x0020_HesaplamalarSoap">
  <soap:address location="http://localhost/geoweb servis/geomat.asmx" />
</port>
</service>

```

WSDL dökümanının içeriği karmaşık gibi görünsede, aslında bu içeriğin oluşturulması ile fazla uğraşmayız. Bir web servisi kullanıcısı, Visual Studio.Net ortamını kullanmasa dahi, bu web servisine ait WSDL dökümanını yazmak zorunda değildir. Sonuç olarak WSDL dökümanı, web servisinden aşağıdaki gibi bir url ile talep edilebilir.

`http://localhost/geowebsevis/geomat.asmx?wsdl`

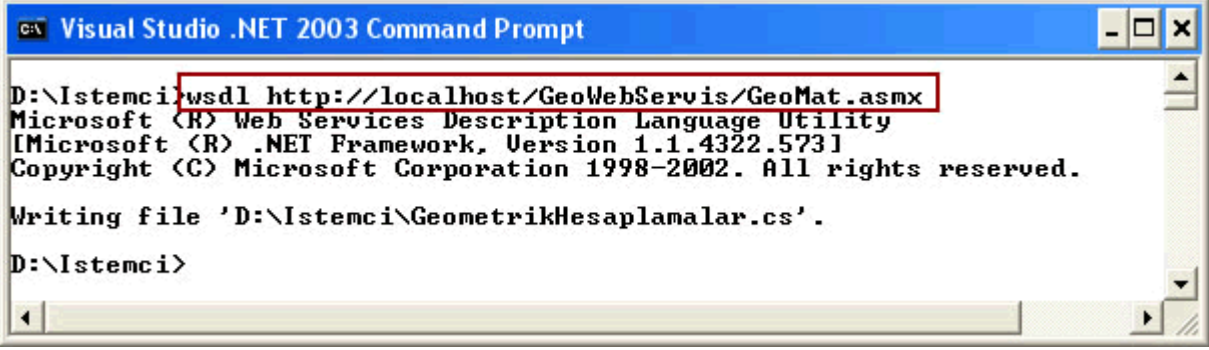
Bu url' nin çalışması sonucunda, yazdığımız web servisi için otomatik olarak bir wsdl dökümanı üretilcektir. Peki elimizde Visual Studio.Net gibi bir geliştirme ortamı yok ise, bu WSDL dökümanını kullanarak bir proxy sınıfını nasıl oluşturabiliriz? Bunun için, .Net framework'te yer alan wsdl aracını kullanmamız yeterlidir.

Buraya kadar işlediklerimiz ile, bir web servisini kullanmak için, bu web servisinin modelini istemci bilgisayarda oluşturacak bir proxy sınıfına ihtiyacımız olduğunu biliyoruz. Bununla birlikte, bu proxy nesnesini oluşturmak için, istemcinin web servisine ait wsdl dökümanını talep ettiğininide biliyoruz. Her ne kadar Visual Studio.Net gibi bir görsel geliştirme ortamında, web servisine ait referans bilgisini istemci uygulamaya bir kaç basit adımda ekleyerek, proxy nesnesinin veri modelini kapsülleyecek sınıfı oluşturmak kolay olsada, bu işi .net framework ile gelen Wsdl aracı yardımıyla da gerçekleştirebiliriz. Sonuç olarak, her zaman elimizin altında Visual Studio.Net gibi profesyonel bir yazılım geliştirme platformu olmayabilir.

Wsdl aracı .Net Framework ile birlikte standart olarak gelen bir araçtır. Görevi, istemci uygulamalar için proxy nesnelere veri modeli sağlayacak proxy sınıflarını oluşturmaktır. Wsdl aracının çeşitli kullanım versiyonları vardır. Bunlardan en çok kullanılanı, web servisine ait asmx dosyasının tam url bilgisini parametre olarak alan aşağıdaki yöntemdir.

Wsdl url/<web servisi adı>.asmx

Şimdi, geliştirmiş olduğumuz web servisine ait bir proxy sınıfını wsdl aracı ile nasıl oluşturabileceğimize bakalım. Bunun için komut satırında aşağıdaki söz dizimini kullanmamız yeterli olacaktır.



```

C:\ Visual Studio .NET 2003 Command Prompt
D:\Istemci>wsdl http://localhost/GeoWebServis/GeoMat.asmx
Microsoft (R) Web Services Description Language Utility
[Microsoft (R) .NET Framework, Version 1.1.4322.573]
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Writing file 'D:\Istemci\GeometrikHesaplamalar.cs'.
D:\Istemci>

```

Şekil 4. WSDL Aracının kullanılması.

Wsdl aracının bu şekildeki kullanımında dikkat etmemiz gereken tek nokta, web servisine ait url bilgisinin tam olarak girilmesidir. Bu işlemin sonucunda, wsdl aracını çalıştırdığımız klasörde, parametre olarak verilen web servise ait bir sınıf dosyası oluşturulduğunu görürüz. Bu dosya, proxy sınıfımıza ait veri modelini kapsülleyen bir yapıda teşkil edilmiştir. Dosyanın adı varsayılan olarak, web servisini oluşturduğumuz sınıf dosyası içindeki, WebServices niteliğindeki Name özelliğininin değerinden alınmıştır.



Şekil 5. Proxy sınıfımız.

İstemci uygulamamız web servisi ile bu proxy sınıfından örneklenecek nesneler üzerinden konuşabilecektir. Wsdl aracı, bir web servisi için bir proxy sınıfını oluştururken, ilk olarak servisten wsdl dökümanını talep eder. Bu işlem sonucunda istemci bilgisayara indirilen wsdl dökümanındaki XML bilgilerinden yararlanılarak, wsdl aracı uygun sınıf kodlarını otomatik olarak oluşturur.

Oluşturulan proxy sınıfına ait kodlar aşağıdaki gibidir.

```
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Web.Services;
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="Geometrik HesaplamalarSoap", Namespace="http://ilk/servis/")]
public class GeometrikHesaplamalar : System.Web.Services.Protocols.SoapHttpClientProtocol
{
    public GeometrikHesaplamalar()
    {
        this.Url = "http://localhost/GeoWebServis/GeoMat.asmx";
    }
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://ilk/servis/DaireAlan",
RequestNamespace="http://ilk/servis/", ResponseNamespace="http://ilk/servis/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public System.Double DaireAlan(System.Double r)
    {
        object[] results = this.Invoke("DaireAlan", new object[] {r});
        return ((System.Double)(results[0]));
    }
    public System.IAsyncResult BeginDaireAlan(System.Double r, System.AsyncCallback callback, object asyncState)
    {
        return this.BeginInvoke("DaireAlan", new object[] {r}, callback, asyncState);
    }
    public System.Double EndDaireAlan(System.IAsyncResult asyncResult)
    {
        object[] results = this.EndInvoke(asyncResult);
        return ((System.Double)(results[0]));
    }
}
```

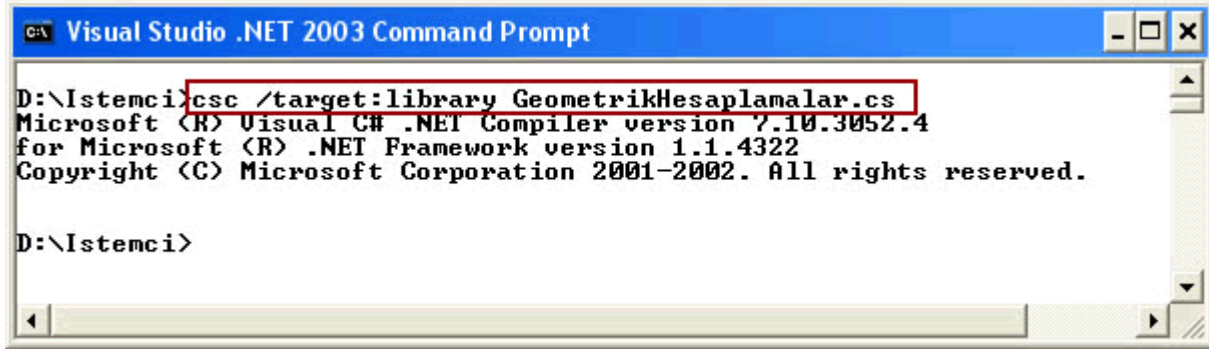
```

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://ilk/servis/DaireCevre",
RequestNamespace="http://ilk/servis/", ResponseNamespace="http://ilk/servis/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
public System.Double DaireCevre(System.Double r)
{
    object[] results = this.Invoke("DaireCevre", new object[] {r});
    return ((System.Double)(results[0]));
}
public System.IAsyncResult BeginDaireCevre(System.Double r, System.AsyncCallback callback, object asyncState)
{
    return this.BeginInvoke("DaireCevre", new object[] {r}, callback, asyncState);
}
public System.Double EndDaireCevre(System.IAsyncResult asyncResult)
{
    object[] results = this.EndInvoke(asyncResult);
    return ((System.Double)(results[0]));
}
[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://ilk/servis/DaireCevreDizi",
RequestNamespace="http://ilk/servis/", ResponseNamespace="http://ilk/servis/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
public System.Double[] DaireCevreDizi(System.Double[] r)
{
    object[] results = this.Invoke("DaireCevreDizi", new object[] {r});
    return ((System.Double[])(results[0]));
}
public System.IAsyncResult BeginDaireCevreDizi(System.Double[] r, System.AsyncCallback callback, object asyncState)
{
    return this.BeginInvoke("DaireCevreDizi", new object[] {r}, callback, asyncState);
}
public System.Double[] EndDaireCevreDizi(System.IAsyncResult asyncResult)
{
    object[] results = this.EndInvoke(asyncResult);
    return ((System.Double[])(results[0]));
}
}

```

Gelelim, bu sınıfı herhangi bir uygulamada nasıl kullanacağımıza. Artık elimizde bir proxy sınıfı olduğu için bu sınıftan faydalanarak herhangi bir uygulamadan web servisine ait metodlara erişebilir ve sonuçları değerlendirebiliriz. Bunu için ilk olarak proxy sınıfımıza

ait bir dll oluşturalım. Bu sayede proxy sınıfını temsil edecek bir assembly'a sahip olmuş olacağız. Böylece, herhangi bir platformdaki programda bu dll dosyasını kullanarak web servisimiz ile konuşabileceğiz. Bunun için komut satırından aşağıdaki satırı çalıştırmamız yeterli olacaktır.

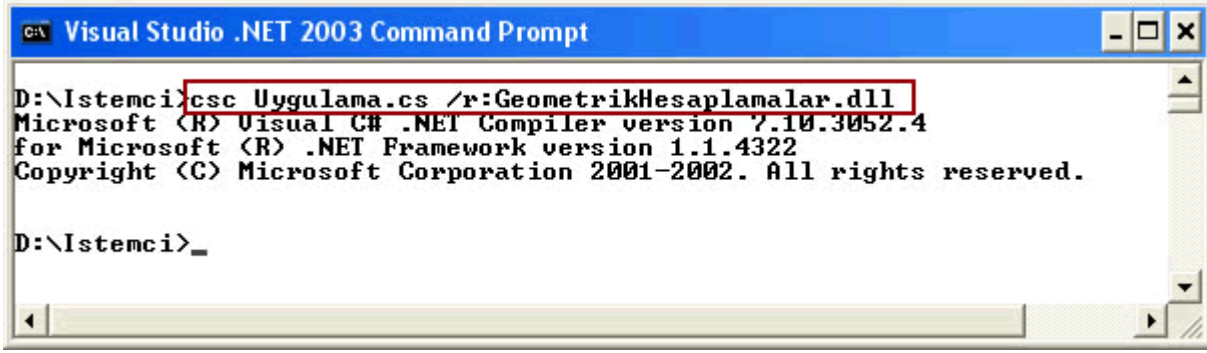


Şekil 6. Proxy sınıfımız için dll' in oluşturulması.

Şimdi bu sınıfı kullanacağımız bir Console uygulaması geliştirebiliriz. Bunun için aşağıdaki kod satırlarını içerecek bir sınıf dosyasını proxy sınıfımıza ait dll ile aynı klasör altında oluşturalım.

```
using System;
using System.Web;
using System.Web.Services;
public class Uygulama
{
    public static void Main(String[] args)
    {
        GeometrikHesaplamalar hesap=new GeometrikHesaplamalar();
        double alan;
        Console.WriteLine("YARICAP ");
        double yariCap=Convert.ToDouble(Console.ReadLine());
        alan=hesap.DaireAlan(yariCap);
        Console.WriteLine("ALAN = {0}",alan);
    }
}
```

Burada dikkat edilecek olursa, proxy sınıfımıza ait bir nesne örneğini oluşturuş şeklimiz, Visual Studio.Net ile geliştirdiğimiz önceki istemci uygulamamızdaki ile aynıdır. Bu benzerlik tesadüfi olmamakla birlikte, web servislerini kullanacak tüm .net uygulamaları için geçerlidir. Kodların yazımı, kullanılan programlama diline göre değişebilir olmasına rağmen teknik olarak yapılan iş, wsdl aracı ile oluşturulan proxy sınıfına ait bir nesne örneğinin oluşturulmasıdır. Örneklendirme işlemini izleyen satırlarda örnek olması açısından, web servisimizdeki DaireAlan metodu çağırılmış ve bu metodun sonucu ekrana yazdırılmıştır. Son aşamada, bu sınıf dosyamızı aşağıdaki şekilde olduğu gibi derlememiz gerekecektir.



```

C:\ Visual Studio .NET 2003 Command Prompt

D:\Istemci>csc Uygulama.cs /r:GeometrikHesaplamalar.dll
Microsoft (R) Visual C# .NET Compiler version 7.10.3052.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

D:\Istemci>_

```

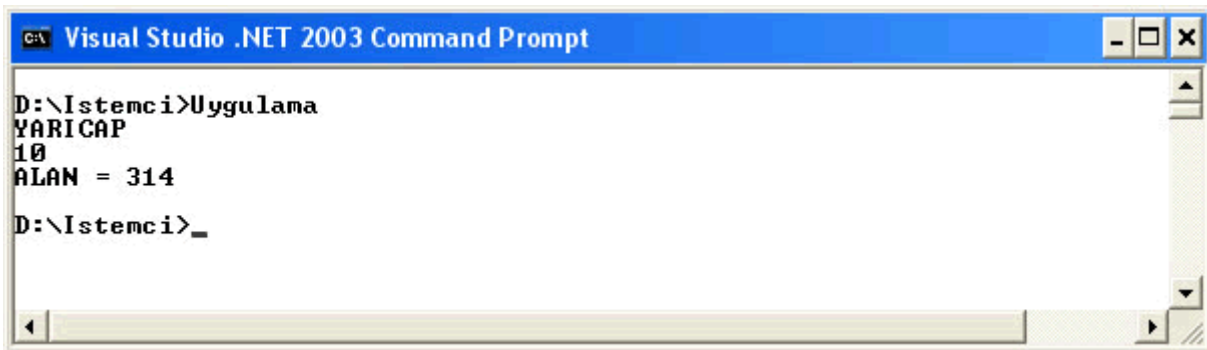
Şekil 7. Uygulamanın çalışması sonucu.

Bu işlemlerin sonucunda uygulamamızın bulunduğu klasördeki dosyalar aşağıdaki gibi olacaktır.



Şekil 8. Dosyalar.

Şimdi uygulamamızı komut satırından çalıştıralım. Ekrandan bir yarıçap değeri girmemiz istenir. Girdiğimiz değer, proxy sınıfından örneklediğimiz nesne aracılığıyla Soap mesajı olarak kodlanarak web servisine iletilir. Web servisi mesajı çözer, parametreyi alır ve ilgili metodu çalıştırır. Sonuç(lar), Soap mesajı olarak kodlanarak, istemciye gönderilir. İstemci uygulamaya ait proxy sınıfınca çözülen mesajdan sonuç(lar) alınır ve konsol ekranına yazdırılır.



```

C:\ Visual Studio .NET 2003 Command Prompt

D:\Istemci>Uygulama
YARICAP
10
ALAN = 314

D:\Istemci>_

```

Son olarak Wsdl aracının kullanımına ilişkin diğer tekniklere bir göz atalım. Proxy sınıfımızın ismini kendimiz belirlemek istersek aşağıdaki örnekte görülen komut satırını kullanırız.

```
Wsdl /out:ProxyAdi.cs http://localhost/geoweb servis/geomat.asmx
```

Bazı durumlarda, web servisinin bulunduğu sunucuya giriş izni gerekebilir. Bunun için wsdl aracını aşağıdaki gibi, domain adı, kullanıcı adı ve şifre parametreleri ile birlikte kullanırız.

Wsdll http://localhost/geowebsevis/geomat.asmx /domain:BURKI /username:Admin /password:Password

Bir web servisinin url bilgisini kullanarak proxy sınıfını oluşturabileceğimiz gibi, fiziki yol bilgisi ile birlikte var olan wsdl belgesini kullanarakta bu sınıfı oluşturabiliriz.

Wsdll d:\inetpub\wwwroot\geowebsevis\geomat.wsdl

Bir proxy sınıfına ait kodlar, standart olarak C# dili baz alınarak oluşturulur. Eğer diğer .net dillerinden birisi ile oluşturulmasını istersek aşağıdaki gibi bir komut satırını kullanırız.

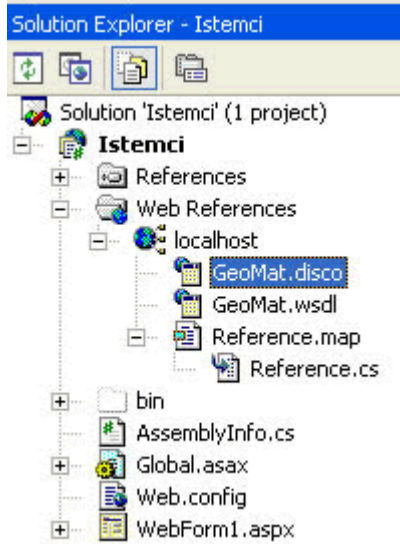
Wsdll /language:VB http://localhost/geowebsevis/geomat.asmx

Bir proxy sınıfını, web servisine ait bir disco dökümanı yardımıyla oluşturabiliriz. Peki disco adı verilen discovery(keşif) dosyalarının web servisleri-istemci sistemindeki yeri nedir? Bu konuyuda bir sonraki makalemizde incelemeye çalışacağız. Tekrardan görüşünceye dek, hepinize mutlu günler dilerim.

Xml Web Servisleri - 5 (Mimarinin Temelleri - DISCO)

Değerli Okurlarım, Merhabalar.

Disco, Microsoft tarafından geliştirilmiş bir keşif mekanizmasıdır. Web servislerinin kullanılması ile ilgili en önemli sorun, istemci uygulamaları geliştiren yazılımcıların, ne tip web servisleri olduğundan ve bunları nasıl kullanacağından haberdar olamamasıdır. Bu amaçla, web servislerini yayınlayanlar, bu servislere ait erişim bilgilerini e-mail veya başka iletişim yolları ile, istemcileri geliştiren yazılım tarafına gönderebilirler. Ancak Microsoft bu işin daha kolay yapılabilmesini sağlamak amacıyla, web servislerinin keşfedilmelerine kolaylık getiren teknikler geliştirmiştir. Bu tekniklerden birisi, disco tekniğidir. Disco tekniğinin kilit noktası, disco uzantılı XML tabanlı dosyalardır. Daha önceki makalelerimizde, Visual Studio.Net ile geliştirdiğimiz istemci uygulamayı göz önüne aldığımızda, GeoMat.disco isimli bir dosyanında yer aldığını görürüz.



Şekil 1. Disco Dosyamız.

Bu dosyanın içeriği XML tabanlı olup aşağıdaki gibidir.

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.xmlsoap.org/disco/">
  <contractRef ref="http://localhost/GeoWebServis/GeoMat.asmx?wsdl" docRef="http://localhost/GeoWebServis/GeoMat.asmx"
xmlns="http://schemas.xmlsoap.org/disco/scl/" />
  <soap address="http://localhost/GeoWebServis/GeoMat.asmx" xmlns:q1="http://ilk/servis/"
binding="q1:Geometrik_x0020_HesaplamalarSoap" xmlns="http://schemas.xmlsoap.org/disco/soap/" /></discovery>
```

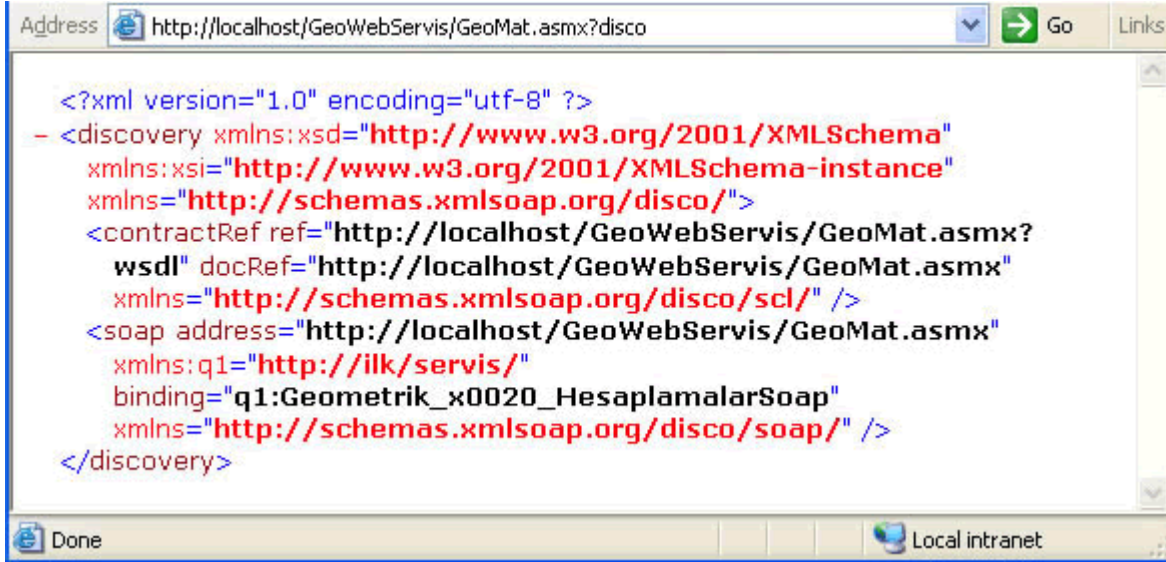
Bu döküman, kullanılan web servisine nasıl erişileceğine ve özellikle wsdl dökümanının nasıl isteneceğine dair önemli bilgileri barındırır. <contractRef boğumunda yer alan ref niteliği ile wsdl dökümanına erişim şekli belirtilir. Aynı boğumda, docRef niteliği ile web servisine ait asmx uzantılı dosyanın bulunduğu url bilgisine yer verilmektedir. Bununla birlikte, <soap boğumunda, soap mesajlaşması için kullanılan isim alanlarının tanımlamaları yer almaktadır.

Bu bilgilere sahip olan bir disco dökümanı sayesinde, kullanılmak istenen web servisine nasıl erişileceği bilinebilir. Disco uzantılı bu XML dosyalarının, web servislerinin keşfine asıl katkısı, .net framework ile gelen disco aracının kullanımında olmaktadır. Bu durumu daha iyi anlayabilmek için şu senaryoyu göz önüne almakta yarar var; bir çok web servisi barındırdan bir web sitemiz olduğunu düşünelim. Web servisleri, IIS gibi sunucularda geliştirildiklerinden, yazılmış olan web servislerinin sayfamızı kullanan istemcilere yayınlamak isteriz. Başka bir deyişle web servislerimizden haberdar olunmasını isteriz. Disco tekniğini bu yayınlama işlemi için kullanabiliriz. İlk olarak, geliştirdiğimiz tüm web servisleri için birer disco dökümanı oluşturmamız gerekir. Daha sonra sitemizin açılış sayfasında ufak bir değişiklik ile bu disco dökümanlarına disco aracı yardımıyla erişilmesini sağlamamız gerekir. Bu noktadan sonra, istemci uygulamaları geliştirecek olan yazılımcıların tek yapması gereken, başlangıç sayfamızın bulunduğu url bilgisini parametre olarak disco aracına bildirmek olacaktır. Şimdi bu senaryoyu gerçeğe çevirecek bir çalışma yapalım.

Örneğin yerel sunucumuzda (localhost) iki web servisimiz olduğunu düşünelim. Şu aşamada bu servislerin nasıl yazıldığına fazla bir önemi yoktur. Şimdi bu servislere ait disco dökümanlarını oluşturmamız gerekiyor. Bir disco dökümanını herhangi bir metin editöründe yazabiliriz. Ancak XML içeriğini oluşturmak zor gelebilir. Bunun için, Asp.Net' in imkanlarından faydalanabiliriz. Asp.Net bir web servisine ait disco dosyasını talep ettiğimizde bu dosyayı browser' da otomatik olarak üretecektir. Bu bir web servisi için wsdl dökümanının istenme şekli ile aynı prensiplere dayanır ve aşağıdaki gibi gerçekleştirilir.

<http://localhost/GeoWebServis/GeoMat.asmx?disco>

Browser' da yazdığımız bu url' den sonra web servisimize ait disco dökümanı bilgilerini aşağıdaki gibi elde ederiz.



Şekil 2. Disco bilgilerinin tarayıcı penceresinden elde edilmesi.

Şimdi tek yapmamız gereken bu XML içeriğini disco uzantılı bir dosya halinde, localhost' a yani inetpub\wwwroot\ klasörü altına kopyalamak olacaktır. Burada dikkat etmemiz gereken bir nokta vardır. Browser' dan elde ettiğimiz xml içeriğini bir disco dosyasına kopylarken – işaretlerini kaldırmayı unutmamalıyız. Aksi takdirde disco aracını kullanırken hata mesajları ile karşılaşırız. Örneğimizde bu ilk disco dökümanını localhost' a GeoMat.disco ismi ile kaydedelim. Aynı işlemi sunucumuzda yer alan tüm web servisleri için gerçekleştirebiliriz. Bu işlemlerin ardından, localhost' un açılış sayfası olan localstart.asp dosyasını herhangi bir editor' de açalım. Burada Head tagları arasına aşağıdaki ekelemeleri yapmamız gerekiyor.

```
<head>
<link type='text/xml' rel='alternate' href='GeoMat.disco'/>
```

Şimdi burada ufak bir işlem daha yapmamız gerekiyor. Head tagında yer alan bu satırdan sadece bir tane tanımlayabiliriz. Normalde aşağıdaki gibi bir tanımlama yapıldığında, belirtilen iki disco dosyasınında ele alınacağı ve istemcide çalıştırılan disco aracı sayesinde indirilebileceğini düşünebiliriz.

```
<link type='text/xml' rel='alternate' href='GeoMat.disco'/>
<link type='text/xml' rel='alternate' href='PerSor.disco'/>
```

Ancak sanılanın aksine, disco aracı belirtilen sayfadaki link taglarından sadece ilkinin çalıştırır ve bu link ile belirtilen disco dökümanını ve buna bağlı wsdl dökümanını indirir. Bu sorunu halletmek için, ilk olarak belirttiğimiz GeoMat.disco dosyasında <discoveryRef bağumunu ref niteliği ile birlikte eklememiz yeterli olacaktır.

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.xmlsoap.org/disco/">
<discoveryRef ref="PerSor.disco"/>
<contractRef ref="http://localhost/GeoWebServis/GeoMat.asmx?wsdl" docRef="http://localhost/GeoWebServis/GeoMat.asmx"
```

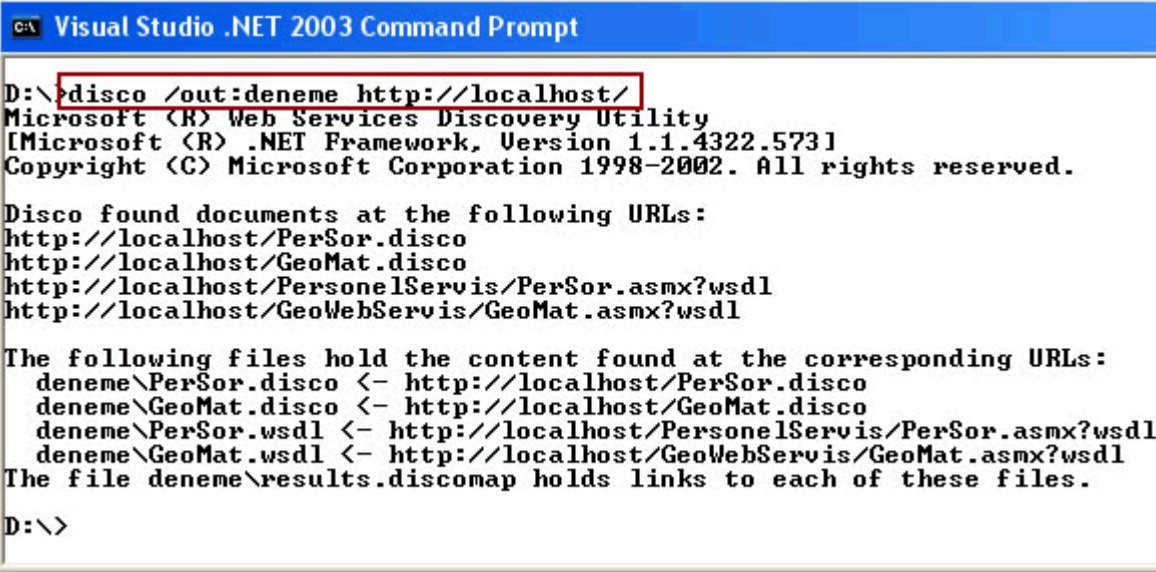
```

xmlns="http://schemas.xmlsoap.org/disco/scl/" />
  <soap address="http://localhost/GeoWebServis/GeoMat.asmx" xmlns:q1="http://ilk/servis/"
binding="q1:Geometrik_x0020_HesaplamalarSoap" xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>

```

DiscoveryRef boğumu ile, GeoMat. disco dosyasının referans edebileceği başka disco dosyalarını belirtmiş oluruz. Dolayısıyla, başlangıç sayfasının bulunduğu adres, disco aracı için parametre olduğunda, bu adresteki link tagında yer alan disco dosyasının yanında bu dosyanın referans ettiği diğer disco dökümanlarında ele alınabilecektir. Açıkçası küçük bir hile yapmış oluyoruz. Şimdi tek yapmamız gereken istemci bilgisayarın komut satırında aşağıdaki ifadeyi çalıştırmak olacaktır.

D:\disco /out:deneme http://localhost/



```

C:\ Visual Studio .NET 2003 Command Prompt

D:\> disco /out:deneme http://localhost/
Microsoft (R) Web Services Discovery Utility
[Microsoft (R) .NET Framework, Version 1.1.4322.5731]
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Disco found documents at the following URLs:
http://localhost/PerSor.disco
http://localhost/GeoMat.disco
http://localhost/PersonelServis/PerSor.asmx?wsdl
http://localhost/GeoWebServis/GeoMat.asmx?wsdl

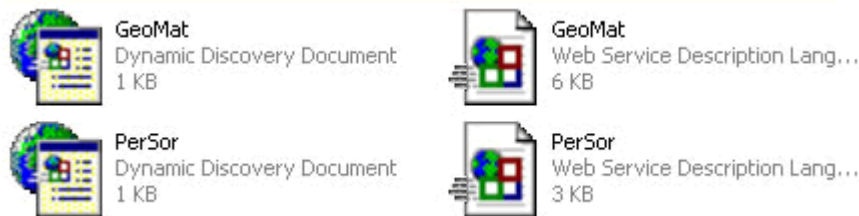
The following files hold the content found at the corresponding URLs:
deneme\PerSor.disco <- http://localhost/PerSor.disco
deneme\GeoMat.disco <- http://localhost/GeoMat.disco
deneme\PerSor.wsdl <- http://localhost/PersonelServis/PerSor.asmx?wsdl
deneme\GeoMat.wsdl <- http://localhost/GeoWebServis/GeoMat.asmx?wsdl
The file deneme\results.discomap holds links to each of these files.

D:\>

```

Şekil 3. Yerel sunucudaki disco ve wsdl dökümanlarının elde edilmesi.

Bu işlemin ardından, deneme isimli klasöre bakıldığında, GeoMat ve PerSor web servislerini kullanabilmemiz için gerekli proxy sınıflarını oluşturmakta kullanacağımız wsdl dökümanlarının disco dökümanları ile birlikte indirildiğini ve kayıt edildiğini görürüz.



Şekil 4. İndirilen disco ve wsdl dökümanları.

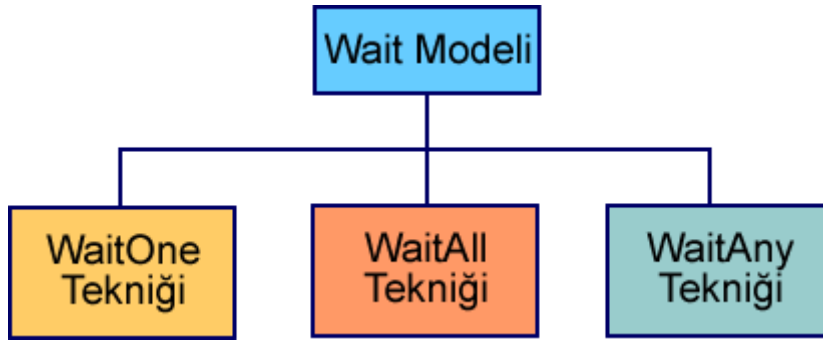
Artık bu noktadan sonra yapmamız gerekenler sadece wsdl dökümanları yardımıyla proxy sınıfımızı oluşturmak ve kullanmak olacaktır. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Ado.Net 2.0 ve Sql Komutlarını Asenkron Olarak Yürütmek – 3

Değerli Okurlarım Merhabalar,

Hatırlayacağınız gibi, Asenkron erişim teknikleri ile ilgili önceki makalelerimizde Polling ve Callback modellerini incelemiştik. Bu makalemizde ise, Wait modelini incelemeye çalışacağız. Wait modeli, diğer asenkron sql komutu yürütme tekniklerine göre biraz daha farklı bir işleyişe sahiptir. Bu model, bazı durumlarda asenkron olarak çalışan sql komutları tamamlanıncaya kadar uygulamayı uyutmak istediğimiz durumlarda kullanılmaktadır. Öyleki, bazen asenkron olarak çalıştırdığımız birden fazla komut birbirlerini etkileyen bir yapıda olabilir. Dolayısıyla bu süreç içinde, sql komutları her ne kadar asenkron olarak çalışsalarda, kesin sonuçları diğer komutlarda etkileyebileceği veya uygulamanın gidiş hatına yön verebileceği için Wait modelinin tercih edileceği durumlar oluşabilir.

Wait modeli şu an için 3 teknik ile gerçekleştirilmektedir. Dilerseniz bu tekniklerin ne olduklarını ve nasıl uygulandıklarını kısaca inceleyelim.



Şekil 1. Wait Modelleri.

Wait Modellerini en kolay haliyle anlayabilmek için işe, WaitOne modeli ile başlamakta fayda vardır. WaitOne modeli, sadece tek bir sql komutu için uygulamanın bekletilmesini sağlar. Ne kastettiğimi anlamak için aşağıdaki örneğimizi dikkatle inceleyelim.

```
#region Using directives
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Threading;
```

```
#endregion
```

```
namespace WaitModeli
```

```
{
    class Program
    {
```

```

static void Main(string[] args)
{
    SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI;async=true");
    con.Open();
    SqlCommand cmd = new SqlCommand("UPDATE Production.Product SET ListPrice=ListPrice*1.15", con);

    IAsyncResult res1 = cmd.BeginExecuteNonQuery(); //Komutu asenkron olarak çalıştır.

    WaitHandle wh1 = res1.AsyncWaitHandle; // WaitHandle nesnesini al.
    Console.WriteLine("Herhangibir işlem"); // Bu aralıkta eş zamanlı kodlar çalışır.

    wh1.WaitOne(); // Eğer komut tamamlanmamışsa bekle.
    int sonucUpdate = cmd.EndExecuteNonQuery(res1); // Komut işleyişini tamamladı sonuçları al.
    Console.WriteLine(sonucUpdate.ToString() + " SATIR GUNCELLENDİ");

    Console.ReadLine();
}
}
}

```

Bu örnek console uygulamasında, Yukon üzerinde yer alan AdventureWorks veritabanındaki Product isimli tabloda güncelleme işlemi gerçekleştiren bir komut söz konusudur. Biz komutu asenkron olarak yürütmek istiyoruz. Lakin komutun sonuçlarını almadan önce bir takım kod satırlarında çalıştırılmasını istiyoruz. Buraya kadar her şey normal. Ancak eş zamanlı yürüyen kod satırlarından sonrada, eğer işlenen komut halen daha tamamlanmamışsa, o komut tamamlanıncaya kadar güncel uygulamanın duraksamasını ve başka bir işlem yapmamasını istiyoruz. İşte bu gibi tek komutlara wait modelini uygulamak istediğimizde, WaitHandle sınıfının statik metodlarından olan WaitOne' ı kullanmaktayız.

Uygulama kodlarımızı inceldiğimizde ilk olarak asenkron olarak çalışmasını istediğimiz komuta ait Begin metodlarından birisini kullanarak IAsyncResult arayüzü tipinden bir nesne örneğini elde ediyoruz.

```
IAsyncResult res1 = cmd.BeginExecuteNonQuery();
```

Daha sonra ise, bu arayüz nesnesinin AsyncWaitHandle özelliği ile bir WaitHandle nesnesi oluşturuyoruz.

```
WaitHandle wh1 = res1.AsyncWaitHandle;
```

Bu WaitHandle nesnesinin elde edilişi sırasında ve sonrasındaki uygulama satırları sql komutumuz ile eşzamanlı olarak yürütülmektedir. Biz WaitHandle nesnemizin WaitOne metodunu uyguladığımızda, çalışan sql komutunun tamamlandığına dair bir sinyal WaitHandle nesnesine gelinceye kadar aktif thread' in duraksatılmasını sağlamış oluyoruz.

```
wh1.WaitOne();
```

Gelelim, WaitAll tekniğine. WaitAll ise, birden fazla sql komutunun asenkron olarak çalıştığı durumlarda devreye giren ve tüm komutlar başarılı bir şekilde tamamlanincaya kadar aktif uygulamayı duraksatan bir tekniktir. Bu sefer, WaitHandle nesneleri her bir komut için ayrı ayrı tanımlanmalıdır. Dolayısıyla bize WaitHandle tipinden bir dizi gerekmektedir. Aşağıdaki örnek uygulama, WaitAll tekniğinin nasıl uygulandığını göstermektedir.

```
SqlConnection con = new SqlConnection("data source=localhost;initial catalog=AdventureWorks;integrated
security=SSPI;async=true");
con.Open();
SqlCommand cmd = new SqlCommand("UPDATE Production.Product SET ListPrice=ListPrice*1.15", con);
SqlCommand cmd2 = new SqlCommand("SELECT * From Person.Address", con);
SqlCommand cmd3 = new SqlCommand("UPDATE Production.Product SET ListPrice=ListPrice*1.20", con);
SqlCommand cmd4 = new SqlCommand("UPDATE Production.Product SET ListPrice=ListPrice*1.45", con);
```

```
IAsyncResult res1 = cmd.BeginExecuteNonQuery();
IAsyncResult res2 = cmd2.BeginExecuteReader();
IAsyncResult res3 = cmd3.BeginExecuteNonQuery();
IAsyncResult res4 = cmd4.BeginExecuteNonQuery();
```

```
WaitHandle[] wh = new WaitHandle[4];
wh[0] = res1.AsyncWaitHandle;
wh[1] = res2.AsyncWaitHandle;
wh[2] = res3.AsyncWaitHandle;
wh[3] = res4.AsyncWaitHandle;
```

```
Console.WriteLine("Burada bir şeyler yapılır...");
for (int i = 1; i < 100; i++)
{
    Console.Write(i.ToString()+" ");
}
Console.WriteLine();
```

```
WaitHandle.WaitAll(wh);
SqlDataReader dr = cmd2.EndExecuteReader(res2);
int sonucGuncel1 = cmd.EndExecuteNonQuery(res1);
int sonucGuncel2 = cmd3.EndExecuteNonQuery(res3);
int sonucGuncel3 = cmd4.EndExecuteNonQuery(res4);
dr.Read();
Console.WriteLine(dr[0].ToString() + " " + dr[1].ToString());
Console.WriteLine(sonucGuncel1 + " SATIR GUNCELLENDİ");
```

```

Console.WriteLine(sonucGuncel2 + " SATIR GUNCELLENDI");
Console.WriteLine(sonucGuncel3 + " SATIR GUNCELLENDI");
Console.ReadLine();

```

Burada görüldüğü gibi, dört adet sql komutumuz var. Her ne kadar anlamsız olsalarda sonuçta amacımız Wait modelinin nasıl işlediğini anlamaktır. WaitHandle dizisi içindeki her eleman, bu elemanlar ile ilişkili komutlara ait IAsyncResult nesnelerinin AsyncWaitHandle özellikleri yardımıyla oluşturulur. Bu esnada, sql komutları asenkron olarak yürütülmektedir. Dolayısıyla WaitHandle sınıfının WaitAll metodunu çalıştırdığımız satıra kadar olan kodlar, komutlar ile birlikte eş zamanlı olarak yürümektedir. WaitAll metodunun olduğu satıra gelindiğinde, uygulama çalışan komutların hepsi tamamlanmış olmak şartıyla yoluna devam eder. Bir başka deyişle, eğer bu satıra gelindiğinde halen daha tamamlanmamış komutlar varsa, varsayılan timeout süresi doluncaya kadar bu komutlarında tamamlanması beklenir.

Wait modelindeki son teknik ise, WaitAny yapısıdır. Bu teknik bir öncekilere nazaran biraz daha karmaşıktır. WaitAny tekniğinde, asenkron olarak yürütülen komutlar sırasıyla tamamlanincaya kadar uygulama bekletilir. Burada sıralamayı belirten WaitAny metodudur. Nitekim, işleyişi önce tamamlanan metodun WaitHandle nesnesi diğerlerine göre daha öncelikli olarak sinyal alacak ve sonuçlar ortama aktarılacaktır. WaitAny tekniğinde işin ilginç ve bir o kadarda önemli olan yanı, WaitAny metodunun geriye integer olarak döndürdüğü değerin kullanılmasında yatmaktadır. Bu değer aslında, WaitHandle dizisi içindeki indeksi işaret etmektedir. Yani, WaitAny metodu ile aktif olarak alınan WaitHandle eğer tamamlanmışsa, metodun geriye döndüreceği değer bu komuta bağlı WaitHandle elemanının dizi içerisindeki indeksidir.

Buna göre, WaitAny tekniğinin, WaitHandle dizisi içindeki tüm elemanları işleyen bir döngü içerisinde kullanılması ve bu metodun döndürdüğü değerlerinde dizi indeksleri ile burada karşılaştırılması gerekmektedir. Her ne kadar karışık bir teknikmiş gibi görünsede, aşağıdaki kod bloğunu incelediğimizde olayı çok daha kolay anlayabileceğimize inanıyorum.

```

WaitHandle[] wh = new WaitHandle[2];
wh[0] = res1.AsyncWaitHandle;
wh[1] = res2.AsyncWaitHandle;
for (int i = 0; i < 2; i++)
{
    // EŞ ZAMANLI İŞLEYEN KODLAR
    int iIndis = WaitHandle.WaitAny(wh);
    if (iIndis == 0)
    {
        cmd.EndExecuteNonQuery(res1);
    }
    if (iIndis == 1)
    {
        SqlDataReader dr = cmd2.EndExecuteReader(res2);
    }
}

```

Dilerseniz bu kod parçasını inceleyelim. WaitHandle dizimizi, iki sql komut için gerekli olan WaitHandle nesnelerini tutmak üzere kullanıyoruz. Bu dizi için gerekli WaitHandle nesnelerini elde edebilmek amacıyla, IAsyncResult arayüzüne ait AsyncWaitHandle özelliğini kullanmaktayız. Buraya kadar herşey anlaşılır. Asıl önemli olan açtığımız for döngüsüdür. Bu döngü, WaitHandle dizisinin eleman sayısı kadar iterasyon yapmaktadır. Döngü içerisinde,

```
int iIndis = WaitHandle.WaitAny(wh);
```

satırı ile, dizi içindeki WaitHandle nesnelerinden her hangibirinin ilgili komutun bittiğine dair bir sinyal alıp almadığına bakılır. Eğer böyle ise, komut işleyişini tamamlamış demektir. Bu durumda WaitAny metodu geriye integer bir değer döndürür. Elbetteki işleyen sql komutlarından her hangibir tamamlanmamış ise, uygulama bu satırda bu komutlardan herhangibiri tamamlanıncaya ve ilgili WaitHandle nesnesine sinyal gönderilinceye kadar duraksar. Dönen integer değer, WaitAny metodunun yapısı gereği, WaitHandle dizisi içindeki ilgili WaitHandle nesnesinin indeks değerini işaret eder. Bu nedenle uygulama bir sonraki satıra geçtiğinde if koşuluna girer ve burada hangi komutun tamamlandığı WaitAny metodundan dönen integer değer yardımıyla belirlenir. Tabiki tamamlanan komuta uygun olan Begin metoduda bu if döngüsü içinde çalıştırılır.

Daha sonra ise, döngü ikinci iterasyondan işleyişine devam eder. Bu kez halen daha çalışmakta olan diğer sql komutlarına ait WaitHandle nesnelerinin bir sinyal alıp almadığına bakılır. Yine, ilk iterasyonda olduğu gibi önce tamamlanan komuta ait WaitHandle nesnesi gerekli sinyali alır ve geriye dizi içindeki indeks değerini döndürür. Ardından tekrar if koşulları uygulanır ve gerekli sonuçlar elde edilir. Bu işleyiş döngü içindeki tüm WaitHandle nesneleri ilişkili oldukları komutların tamamlandığına dair sinyaller alıncaya, dolayısıyla komutlar işleyişini bitirinceye kadar devam eder.

Görüldüğü gibi Wait modeli içerdiği teknikler itibariyle biraz karışıktır. Ancak ilerleyen zamanlarda, Ado.Net 2.0' ın son sürümünde bu tekniklerin çok daha kullanışlı hale geleceğine inanıyorum. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Ado.Net 2.0 ve SqlDependency Sınıfı Yardımıyla Query Notification

Değerli Okurlarım Merhabalar,

Çoğu zaman istemci uygulamalarda, kullanıcıya sunduğumuz verilerin yer aldığı tablolarda başka kullanıcılar tarafından gerçekleştirilen değişikliklerin anında görünmesini isteriz. SqlDependency sınıfı sayesinde artık, bir veritabanında meydana gelen değişiklikleri (şu an için Sql Server ve Yukon) anında yakalayabilme şansına sahibiz. İşte bugünkü makalemizde aslında son derece derin ve geniş bir konu olan Sql Server Notification meselesini çok basit bir örnek ve Ado.Net 2.0 ile gelen yeni sınıflardan birisi olan SqlDependency yardımıyla incelemeye çalışacağız. İlk olarak şu soruyu düşünelim.

Sunucu üzerinde ne gibi gelişmeler olduğunda anında haberdar olmak isteyebiliriz?

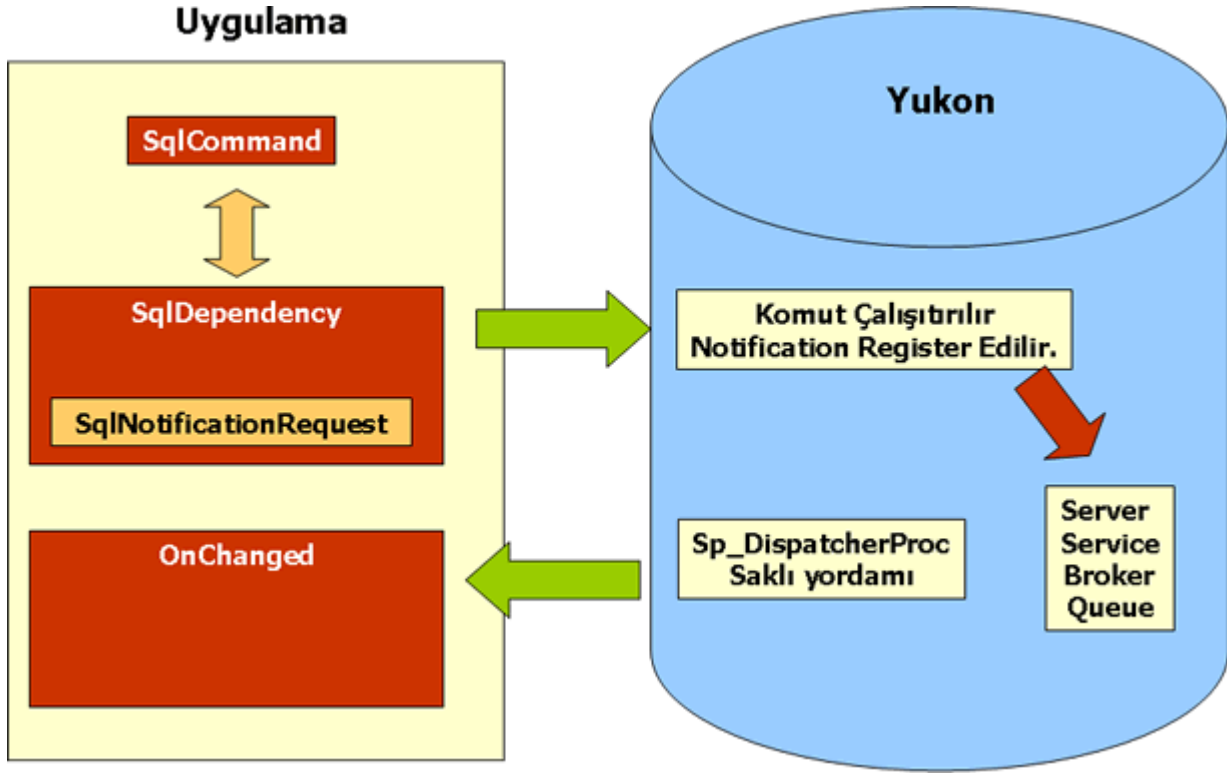
Verilerde meydana gelebilecek değişiklikler; bir başka deyişle yeni satırların eklenmesi, satırların silinmesi veya güncellenmesi.

Tablo üzerinde Drop, Alter veya Delete işlemlerinin uygulanması.

Sql sunucusunun yeniden başlatılması.

Sql sunucusu üzerinde hatalar oluşması nedeniyle sunucunun işleyişinin durması.

İşte bu gibi durumlarda, istemci uygulamaların meydana gelecek olan değişikliklerden anında haberdar olmasını isteyebiliriz. Bu durum özellikle, bağlantısız katman nesnelerinin kullanıldığı uygulamalarda büyük önem kazanmaktadır. Peki bu anında haber bildirilmesini nasıl sağlayabiliriz? Bunun için kullanılan mimari kabaca aşağıdaki şekilde olduğu gibidir.



Şekil 1. Mimari.

Dilerseniz kısaca mimarinin işleyişinden ve bileşenlerinden bahsedelim. SqlDependency sınıfı, bir SqlCommand ile ilişkilendirilerek kullanılır ve içsel olarak bir SqlNotificationRequest nesnesinde barındırır. (Bu sınıfı ilerleyen makalelerimizde incelemeye çalışacağız.) SqlCommand sınıfının Notification işlemine uyan bir Select sorgusu içermesi gerekir. Öyleki bu Select sorgusu şu anki versiyonlar itibarıyla aşağıdaki gibi olamaz.

```
Select * From Tablo
```

Select sorgusunda talep edilen alan isimleri açıkça belirtilmelidir. Bununla birlikte, Tablo' nun sahibide yani yaratıcısıda açıkça belirtilmelidir. Yani ancak aşağıdaki gibi bir sql ifadesi Notification desteğini sağlayabilir.

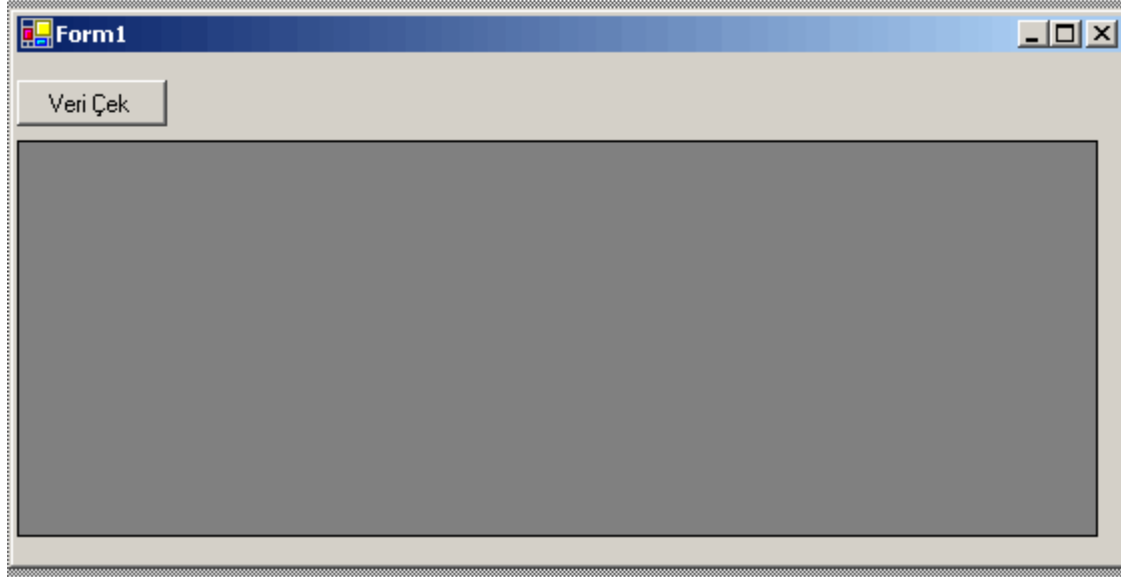
```
Select Ad,Soyad From dbo.Tablo
```

Belirtilen Sql ifadesi çalıştırıldığında, SqlDependency nesnesi ilgili SqlCommand nesnesi için sunucuda bir Notification açılacağını sql sunucusuna bildirir. Sql sunucusu bu mesaj üzerine ilgili sql komutunu çalıştırır ve Notification nesnesini sunucuya register eder. Bu bize, çalıştırılan sql sorgusunun sunucuya ekstradan bir takım bilgiler daha gönderdiğini göstermektedir. Bu bilgiler çoğunlukla, kaydı tutulacak Notification ile ilgilidir ve SqlDependency sınıfının ilgili yapıcı metodları yardımıyla ayarlanabilir.

Notification nesnesi, sunucuya kayıt edildikten sonra, komutun çalışması ile elde edilen sonuçlar ön belleğe alınır ve ilgili Notification nesneside Server Service Broker Queue tarafından kuyruğa atılır. Buradaki ön bellekleme işlemi aslında bir temp

tablosunun oluşturulmasından başka bir şey değildir. Bu noktadan itibaren Sql sunucusu, ilgili satır kümesini izlemeye başlar. Eğer satır kümesi üzerinde yukarıda bahsettiğimiz durumlardan birisi nedeni ile değişiklikler olursa, sp_DispatcherProc sistem prosedürü yardımıyla istemci uygulamanın SqlDependency sınıfına ait olan OnChanged olayına bilgi mesajı gider.

Mimari her ne kadar karışık gibi görünsede, uygulanabilirliği son derece kolay ve basittir. Bu amaçla olayı daha iyi irdeleyebilmek için aşağıdaki Windows uygulamasını göz önüne alalım.



Şekil 2. Windows Uygulamamız

#region Using directives

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using System.Data.SqlClient;
```

#endregion

```
namespace NotifySample
{
    partial class Form1 : Form
    {
        public Form1()
        {
```

```

        InitializeComponent();
    }
    SqlConnection con;
    SqlCommand cmd;
    SqlDependency dep;
    SqlDataAdapter da;
    DataTable dt;

    private void btnVeriCek_Click(object sender, EventArgs e)
    {
        con = new SqlConnection("data source=localhost;initial catalog=Dukkan;integrated security=SSPI");
        con.Open();
        cmd = new SqlCommand("Select DetayID,PersonelID,Ad,Soyad,Mail From dbo.PersonelDetay", con);
        dep = new SqlDependency(cmd);
        dep.OnChanged+=new OnChangedEventHandler(dep_OnChanged);
        da = new SqlDataAdapter(cmd);
        dt = new DataTable();
        da.Fill(dt);
        dgVeriler.DataSource = dt;
    }

    void dep_OnChanged(object sender, SqlNotificationEventArgs e)
    {
        MessageBox.Show("YENI VERILER EKLENDI...TAZELEME YAPIN");
    }
}

```

Uygulamamız, Yukon üzerinde Dukkan isimle veritabanında yer alan PersonelDetay isimli tabloya ait verileri kullanmaktadır. Kullanıcı Veri Çek başlıklı butona bastığında ilgili tabloya ait tüm satırlar DataGridView kontrolüne yüklenmektedir. Şimdi gelelim, burada önemli olan kod satırlarına. Dikkat ederseniz, SqlDependency sınıfına ait nesne örneğimizi aşağıdaki kod satırı ile oluşturuyoruz. Böylece ilgili komut için Notification servisinin kullanılacağını belirtmiş oluyoruz.

```
dep = new SqlDependency(cmd);
```

Bu arada Select sorgumuzun kurallara uygun bir yapıda olduğuna dikkat edelim. Yani, alan adlarını ve tablonun sahibini açıkça belirtiyoruz. Daha sonra ise, SqlDependency sınıfımıza OnChanged olayını aşağıdaki kod satırı ile ekliyoruz.

```
dep.OnChanged+=new OnChangedEventHandler(dep_OnChanged);
```


Bu olayın bizim için büyük bir önemi var. Nitekim, sunucu üzerinde ön belleklenen veri kümesinde değişiklikler olduğunda, sql sunucusu tarafından uygulamaya bir Notification mesajı gönderilecektir. İşte bu mesajın ele alınacağı olay SqlDependency sınıfına ait OnChanged' dir. Bu olayın prototipi aşağıdaki gibidir.

```
public event OnChangedEventHandler OnChanged;
```

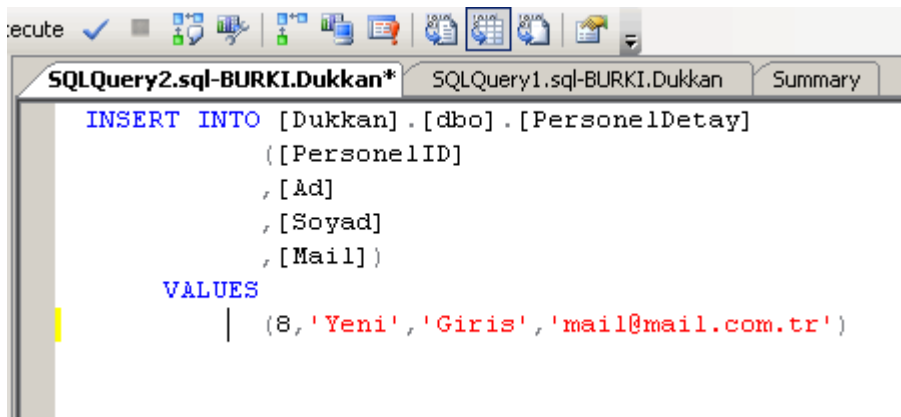
Gördüğümüz gibi bu olay OnChangedEventHandler temsilcisinin tanımladığı metodları çalıştırabilir. OnChangedEventHandler temsilcisinin prototipi ise aşağıdaki gibidir.

```
public sealed delegate void OnChangedEventHandler( object sender, SqlNotificationEventArgs e );
```

Bu temsilcinin işaret edeceği metodlara ait ikinci parametre SqlNotificationEventArgs sınıfı tipinden olup oldukça önemlidir. Nitekim bu parametre yardımıyla, gelen bildirinin hangi olaya istinaden oluştuğunu ayırt edebilir ve işlemlerimize yön verebiliriz. Bu sınıfta detaylı bir şekilde ilerleyen makalelerimizde inceleyeceğiz. Olay metodumuzda ise basit olarak herhangi bir Notification mesajı alındığında bir MessageBox' ın çıkmasını sağlıyoruz.

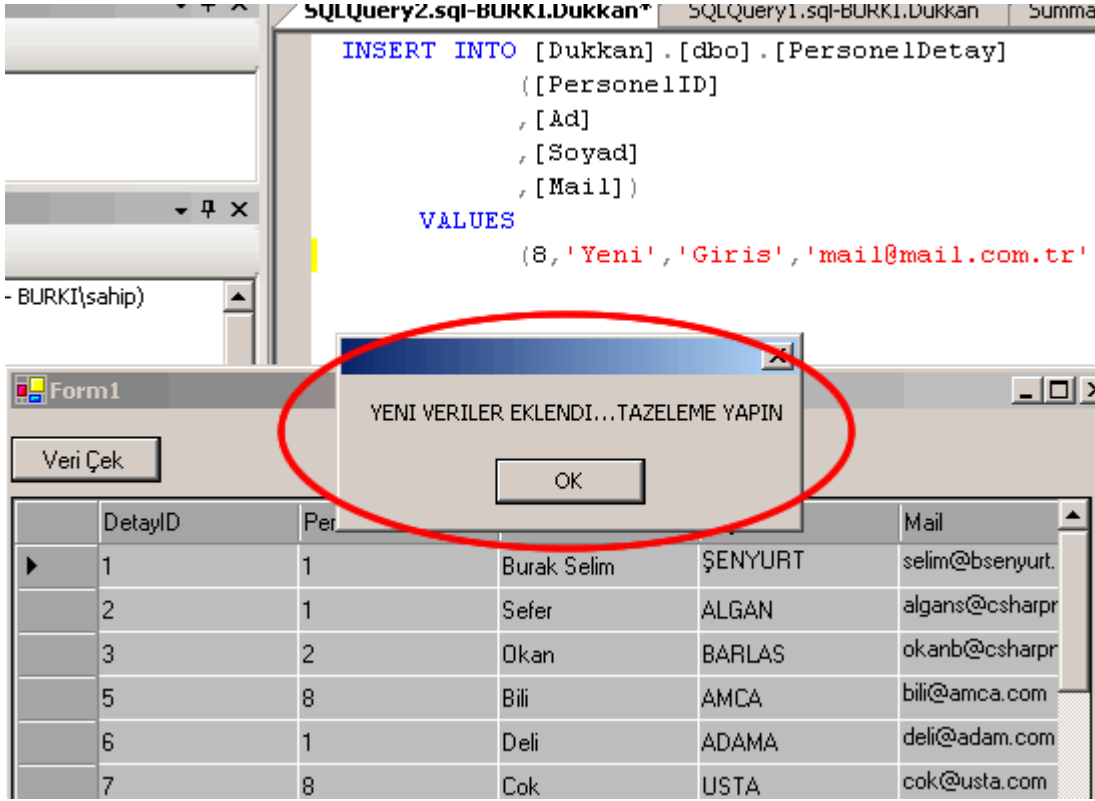
```
void dep_OnChanged(object sender, SqlNotificationEventArgs e)
{
    MessageBox.Show("YENI VERILER EKLENDI...TAZELEME YAPIN");
}
```

Şimdi uygulamamızı çalıştıralım ve verileri çekelim. Yazdığımız kodlar gereği, Sql suncusu üzerinde Select sorgumuza istinaden bir Notification nesnesi register edilecektir. Şimdi biz bu noktada doğrudan Yukon üzerinden yeni bir satır gireceğiz. (Aslında aynı uygulamanın başka bir örneğinden veya başka bir uygulamada bu tablo üzerinde değişiklikler yapabiliriz) Bunu uygulamamız çalışırken yapacağız.



Şekil 3. Yukon üzerinden tablomuza yeni bir satır ekliyoruz.

Bu işlemi gerçekleştirdiğimiz anda, uygulamamıza bir bildiri mesajı iletilir ve tabloda değişiklik olduğu söylenir. Bunun sonucu olarakta, SqlDependency nesnemiz ile ilişkilendirdiğimiz OnChanged olayı aktif hale gelecektir. Sonuç aşağıdaki gibi olacaktır. Yeni satır eklenir eklenmez uygulama OnChanged olayını çalıştıracaktır.



Şekil 4. Notification mesajı uygulamaya gönderildi.

Dolayısıyla biz, veri çekme işlemimizi bu olay içerisine alarak istemci uygulamanın daima güncel veriler ile konuşmasını sağlayabiliriz. SqlDependency sınıfının pek çok üyesi ve önemli özelliği vardır. Bir sonraki makalemizde, bu sınıfı daha detaylı bir şekilde incelemeye çalışırken, gerçekleşen değişikliğin tipine göre nasıl hareket edebileceğimize göreceğiz. Tekrardan görüşünceye dek sağlıklı kalın.

Oyun Programlamaya Giriş (Çarpışma Teknikleri - 1)

Değerli Okurlarım Merhabalar,

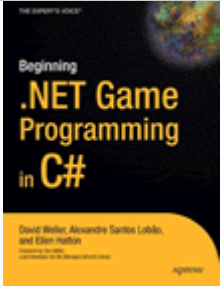
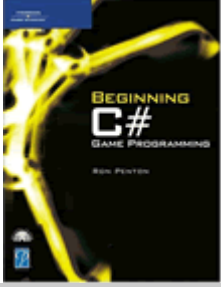
Yaklaşık bir ay kadar önce evde dinlenirken, şu ana kadar yaptığım işleri ve projeleri düşündüm. Kesin olarak şunu söyleyebilirim ki, profesyonel anlamda ilgilendiğim ve kullandığım tek dil C# idi. C# dilini kullanarak, .Net platformu altında veritabanı ağırlıklı olmak üzere çeşitli çalışmalar yaptım. Ancak bir süre sonra farkettim ki, bir Matematik Mühendisi olarak lisans eğitimim sırasında gördüğüm o devasa cebir problemleri, teorem ispatları hiç bir zaman işin içine girmemişti. Matematiğin belkide çok az olmakla birlikte dört işleminin ve bir takım algoritmalar için gerekli iteratif ifadelerinin yer aldığı uygulamalar dışında, onu çok yoğun şekilde kullanmamıştım.

Belki finansal veya istatistiki bir projede yoğun olarak ekonomi teoremlerini ve bunlara bağlı olarak matematiksel denklemleri kullanmak gerekmekeydi ancak çok yoğun bir şekilde bunları kurcalamamıştım. Düşündükçe, günümüz dünyasında artık algoritmalar ile, matematik ile uğraşan yazılımcıların azaldığı kanısına vardım. Hepimizde matematik temeli var. Hepimiz bu konuda çeşitli sınavlardan geçtik. Ama sonuç itibarıyla çok az projede, daha önceden gördüğümüz eşsiz teoremleri kullandık. O dakikalarda

neden bu yöne eğilmiyorum, biraz eğlenceli hatta matematik yüklü çalışmalara el atmıyorum diye düşünmeye başladım. Derken çözümü son derece eğlenceli ama bir o kadarda önemli bir sahada buldum. Oyun Programcılığı.

Çoğumuz, yazılım geliştirirken bir gün büyük bir oyunu yazan ekibin içinde olmayı, onun geliştiricilerinden birisi olarak anılmayı hayal etmişizdir. Bu çok çalışmayı ve birazda zeki olmayı gerektiren bir kişilik ister. Kendi kendime düşünürken, elbette günümüzün popüler oyunlarından birisini yazan herhangi bir ekipte olabilmek için çok erken olduğunu zaten biliyordum. Ama en azından bir oyun için gerekli en temel bileşene biraz da olsa aşinalığım vardı. Matematik. Oyun programlama içerisinde, oyun motorlarının geliştirilmesinden aksiyonların uygulanışına, yapay zeka taktiklerinden stratejik karar mekanizmalarına kadar her aşamada Matematiksel algoritmaların yer aldığını gayet iyi biliyordum. Benim için bunları öğrenmeye çalışmak, uygulamak ve denemek, benim için heyecan verici olacaktı. Diğer taraftan eski Matematik günlerimi hatırlamış bir başka deyişle saksıyı biraz daha çalıştırmış olacaktım.

Büyük bir hevesle bir taslak plan hazırladım ve işe koyuldum. Bana öncelikle oyun programlamadaki temel teknikleri başlangıç seviyesinden itibaren anlatacak ve yeri geldiğinde de uzmanlık seviyesine kadar çıkartacak kitaplar gerekiyordu. Hemen Amazon.com' da kısa bir araştırmadan sonra aşağıdaki kitapların siparişlerini verdim.

Oyun Programlama İle İlgili Kaynak Kitaplar	
	Beginning .Net Game Programming in C# APress David Weller, Alexandre Santos Lobao, Elen Hatton 414 Sayfa Amazon Fiyatı : 30.59\$
	C# and Game Programming: A Beginner's Guide AK Peters Ltd. Salvatore A. Buono 592 Sayfa Amazon Fiyatı : 39\$
	Beginning C# Game Programming (Game Development) Muska & Lipman/Premier-Trade Ron Penten 344 Sayfa Amazon Fiyatı : 20.39\$



Managed DirectX 9 Kick Start : Graphics and Game Programming

Sams

Tom Miller

432 Sayfa

Amazon Fiyatı : 23.79 \$

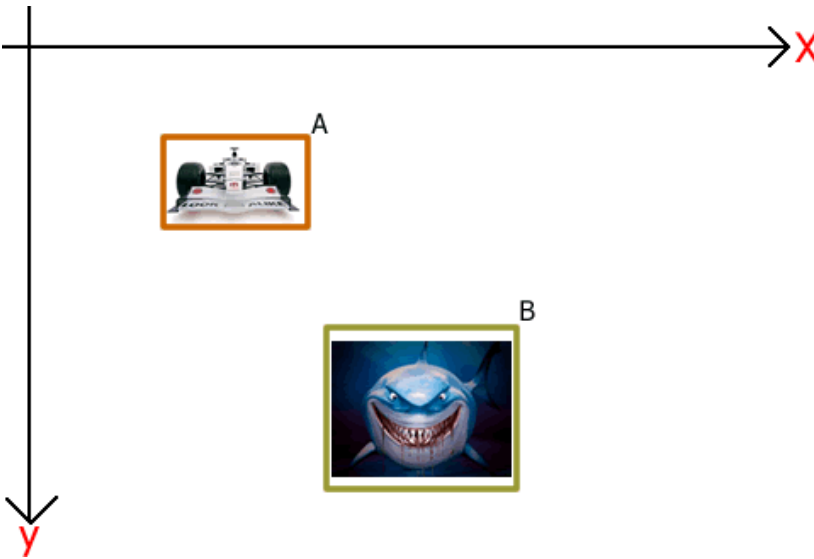
Bu anlattığım olaylar yaklaşık bir ay kadar önce gerçekleşti. Şu anda bu kitaplarda az da olsa ilerlemiş durumdayım. İnanın sevgili okurlarım bu işe giriştiğim için çok ama çok memnunuz. Biraz matematik, biraz teori biraz pratik derken bir şeyler kapmaya başladım bile. Herşeyden önce ilk konularda, eski dostumuz Hipotenüs' ü görmek bile son derece güzeldi. Konuları anlamaya başladığıma göre şimdi tek yapmam gereken öğrendiklerimi uygulayarak pekiştirmek ve sizler ile paylaşmak. Artık bu kadar laf kalabalığından sonra, bu makalemizin konusunada değinme zamanı geldi.

Bu gün, oyun programlamanın önemli temellerden birisi olan Çarpışma (Collision) tekniklerine giriş yapacağız. Bir oyunda, bir birinden bağımsız öğelerin bir birleriyle çarpışmaları üzerinde duracağımız asıl konu olacak. Bir savaş oyununda tarafların bir birlerine karşı yaptıkları hamleler sonucu kimin kime vurduğunu tespit etmek, vuruşların yönüne veya şiddetine göre, darbeyi alan nesnelerin ne tür hareketlerde bulunacağına karar vermek açısından Çarpışma teknikleri gerçekten önemlidir. Bu teknikler bir kaç tanedir. Bu gün ben sizlere, 2 boyutlu koordinat sisteminde dörtgensel şekillerin çarpışmalarının nasıl tespit edilebileceğini anlatmaya çalışacağım.



Dortgenlerin baz alındığı bu teknikte amaç, objeleri içine alan ve sınırlayan dörtgensel alanların birbirleri üstüne gelip gelmediklerinin tespit edilebilmesidir.

Öncelikle durumu analiz edebilmek amacıyla aşağıdaki şekli göz önüne alalım.



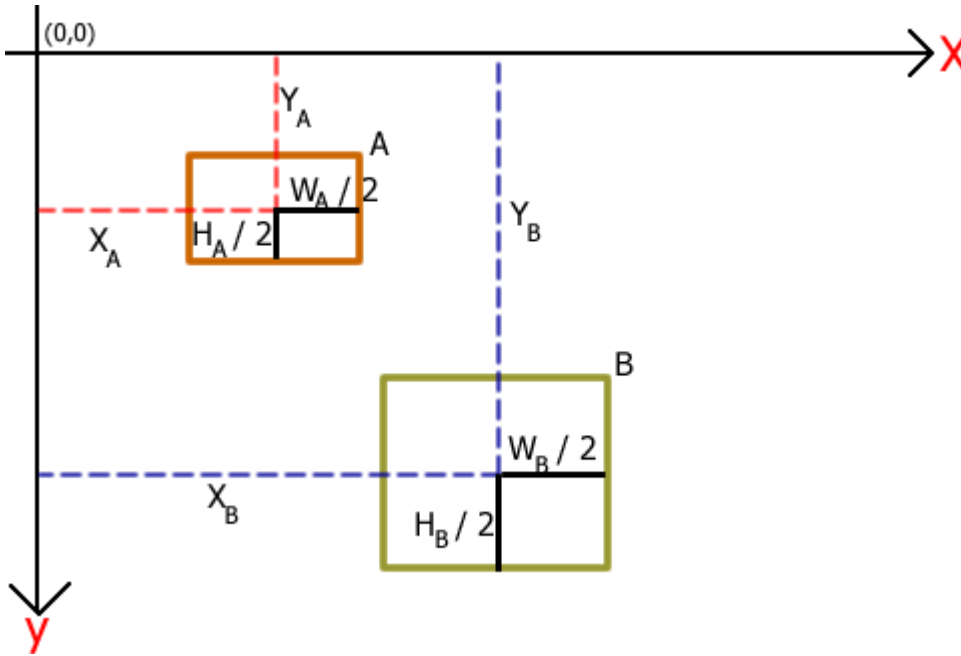
Şekil 1. Senaryo.

Senaryo gereği, köpek balığı ve yarış arabası nesnelerimizin hareketli olduklarını düşünelim. Burada, yarış arabası ve köpek balığı aslında ekranda piksel olarak yer kaplamaktadır. Bu iki piksel topluluğunun çarpışıp çarpışmadıklarını test edebilmek için kullanılabilecek en basit teknik, nesnelerin çevresini saran dörtgenlerin bir birleri üstüne gelip gelmediklerini tespit etmektir. Bu nedenle, senaryomuzda yarış arabamızı A isimli, köpek balığımızı ise B isimli dörtgenler ile çevrelediğimizi düşünelim. İşte bu noktadan sonra işin içine biraz matematik girecek.



Eğer bu iki nesnenin X koordinatları arasındaki fark, genişliklerinin yarılarının toplamından küçük ise ve iki nesnenin Y koordinatları arasındaki fark, yüksekliklerinin yarılarının toplamından küçük ise, bu iki dörtgen üst üste gelmiş dolayısıyla çarpışma (Collision) gerçekleşmiş demektir.

Bunun matematiksel olarak aşağıdaki şekil ile daha iyi anlayabiliriz.



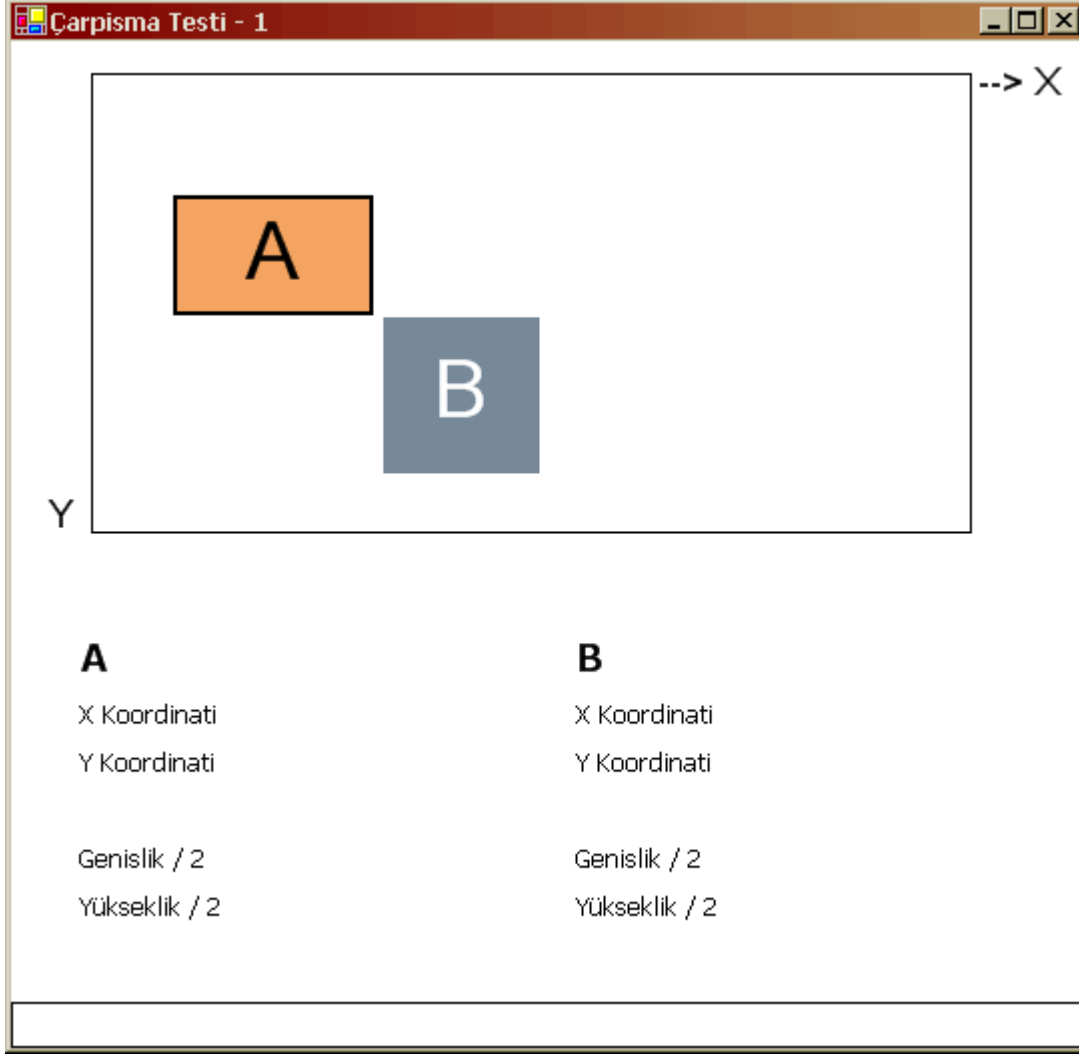
Şekil 2. Koordinat, genişlik ve yükseklik ölçüleri.

Burada, her iki dörtgenin ekranın sol üst köşesine olan uzaklıkları ve genişlik ile yükseklikleri belirtilmektedir. Elbette bir nesnenin X koordinatı uygulamada bu nesnenin Left özelliğinin değerine, Y koordinatı ise Top özelliğinin değerine işaret etmektedir. Aynı şekilde genişlik için Width, yükseklik için ise Height özelliklerinden yararlanılacaktır. Yukarıda bahsettiğimiz çarpışma koşulunun matematiksel ifadesi ise aşağıdaki gibi olacaktır.

$$\begin{aligned} |X_A - X_B| &< W_A / 2 + W_B / 2 \\ \text{VE} \\ |Y_A - Y_B| &< H_A / 2 + H_B / 2 \end{aligned}$$

Şekil 3. Çarpışma (Collision) koşulları.

Dikkat edecek olursanız eşitsizliklerin sol taraflarındaki ifadelerde mutlak değer söz konusudur. Bu nedenle Math sınıfının Abs isimli metodu işimize çok yarayacaktır. Bu ifadedeki her iki eşitsizliğinde gerçekleşmesi halinde, dörtgenlerin üst üste geldiklerinden, dolayısıyla çarpıştıklarından söz edebiliriz. Bu teorimin gerçekliğini kontrol etmenin en güzel yolu uygulama üzerinde olacaktır. Bu amaçla aşağıdaki gibi basit bir windows uygulaması oluşturdum.



Şekil 4. Uygulama.

Bu uygulamada, A isimli dörtgen nesnesini klavyedeki A,S,D,E tuşları ile hareket ettirebileceğiz. Bizim buradaki amacımız, Çarpışma olduğu takdirde bunun gerçekleşip gerçekleşmediğini tespit edebilmek. Bu amaçla, nesnelerin ekrandaki koordinatlarında izlediğimiz label nesnelerimiz var. Gelelim uygulamamızın kodlarına.

```
private void Yaz()
{
    lblXA.Text=A.Left.ToString();
    lblYA.Text=A.Top.ToString();
    lblWidthA.Text=Convert.ToString((A.Width/2));
```

```

lblHeightA.Text=Convert.ToString((A.Height/2));

lblXB.Text=B.Left.ToString();
lblYB.Text=B.Top.ToString();
lblWidthB.Text=Convert.ToString((B.Width/2));
lblHeightB.Text=Convert.ToString((B.Height/2));
}

private void frmCarpisma_Load(object sender, System.EventArgs e)
{
    Yaz();
}

/* Çarpışma Kontrolünün gerçekleştirildiği metod */
private bool CarpismaKontrol()
{
    float mutlakX=Math.Abs((A.Left+(A.Width/2))-(B.Left+(B.Width/2)));
    float mutlakY=Math.Abs((A.Top+(A.Height/2))-(B.Top+(B.Height/2)));

    float farkGenislik=(A.Width/2)+(B.Width/2);
    float farkYukselik=(A.Height/2)+(B.Height/2);

    if((farkGenislik>mutlakX)&&(farkYukselik>mutlakY))
    {
        return true;
    }
    else
        return false;
}

private void frmCarpisma_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
{
    if(e.KeyChar==(Char)Keys.D)
    {
        A.Left+=10;
    }
    if(e.KeyChar==(Char)Keys.A)
    {
        A.Left-=10;
    }
}

```

```

    }
    if(e.KeyChar==(Char)Keys.W)
    {
        A.Top-=10;
    }
    if(e.KeyChar==(Char)Keys.S)
    {
        A.Top+=10;
    }

    Yaz();
    if(CarpismaKontrol())
    {
        lblCarpismaKontrol.Text="Çarpisma var...";
    }
    else
    {
        lblCarpismaKontrol.Text="";
    }
}

```

Uygulamada dikkat ederseniz, teoremi aşağıdaki kod satırlarındaki gibi uyguladık.

```

float mutlakX=Math.Abs((A.Left+(A.Width/2))-(B.Left+(B.Width/2)))/ * X koordinatları arası farkın mutlak değeri. */
float mutlakY=Math.Abs((A.Top+(A.Height/2))-(B.Top+(B.Height/2)))/ * Y koordinatları arası farkın mutlak değeri . */

```

```

float farkGenislik=(A.Width/2)+(B.Width/2); /*Genişliklerin yarılarının toplamı.*/
float farkYukselik=(A.Height/2)+(B.Height/2); /* Yüksekliklerin yarılarının toplamı. */

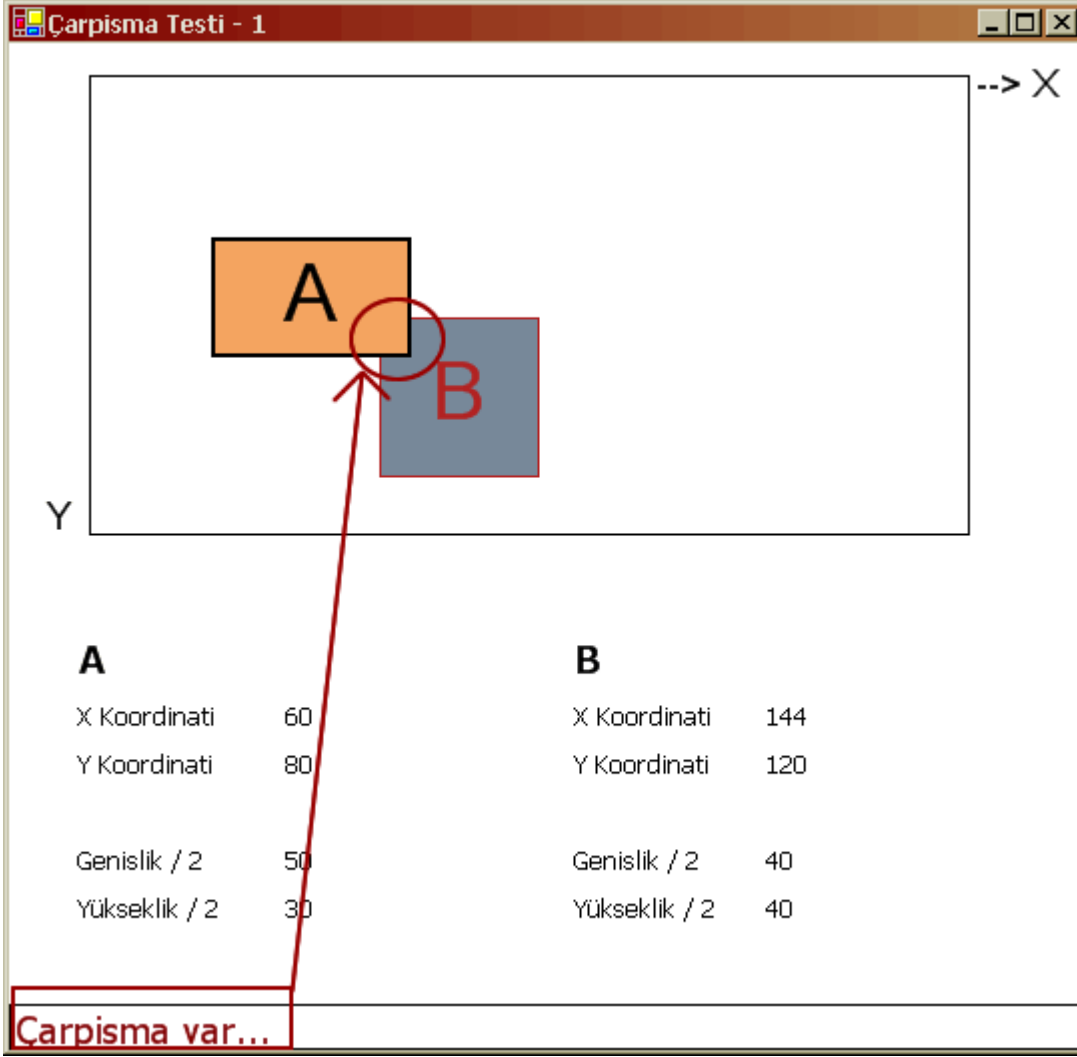
```

```

if((farkGenislik>mutlakX)&&(farkYukselik>mutlakY)) /* Eğer koşul doğru ise çarpışma vardır. */
{
    return true;
}
else
    return false;
}

```

Şimdi uygulamamızı test edelim. A, S, D, W tuşları ile (bunlara basarken Caps Lock açık olmalı) A isimli buton kontrolümüzü 10' ar birim hareket ettirebilmekteyiz. Eğer kutuları üstüste getirirsek çarpışma teoreminin başarılı bir şekilde gerçekleştiğini görürüz.



Şekil 5. Çarpışma gerçekleşti.

Bu teknik en basit çarpışma modelidir. Nesnelerin dörtgenler içerisinde düşünülmesi araba şeklinde olduğu gibi, boşluğa denk düşen alanlarında hesaba katılmasına neden olmaktadır. Ancak iş dairesel nesnelerin çarpışmasına geldiğinde (örneğin tenis toplarının raketler çarpması) hatta, daire ve karesel nesnelerin çarpışmasına geldiğinde dahada karmaşıklaşmaktadır.

Artık ilerleyen zamanlarda bu teoremleri incelemeye çalışacağım. Öğrendikçede siz değerli okurlarıma aktaracağım. Artık 24 bölümümü sürer bir tetris oyunu yazmamız, 24 ay mı sürer bilemiyorum. En azından daha önceden bildiğim hatta mutlaka bildiğim ama farkına varamadığım algoritmaları öğrenmek beni oldukça memnun etti. Umuyorum ki sizlerde bu yazı dizisinden hoşnut kalırsınız. Tekrar görüşünceye dek hepinize mutlu günler dilerim.

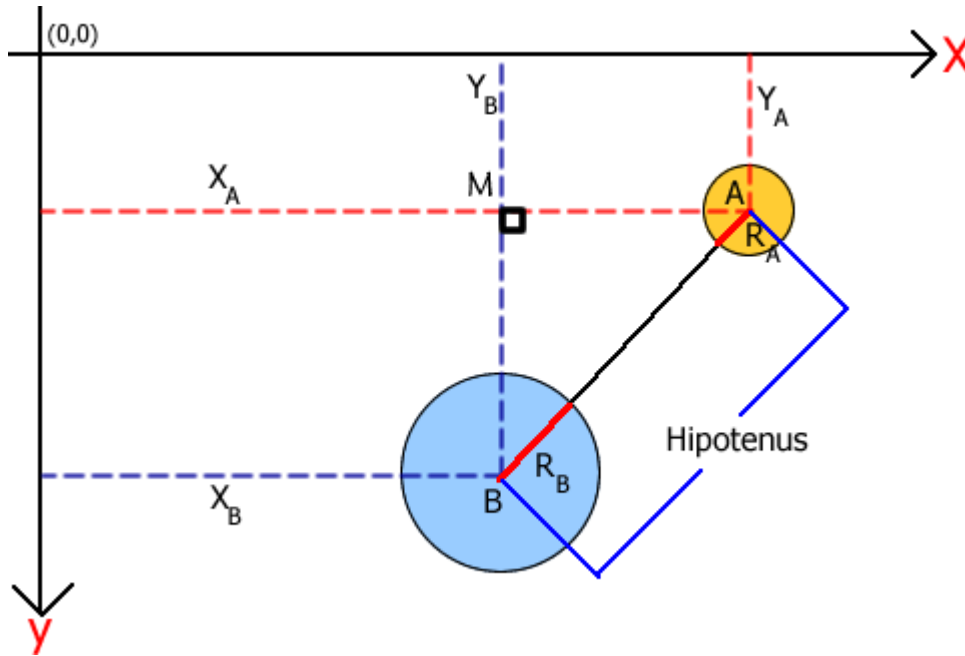
Oyun Programlamaya Giriş (Çarpışma Teknikleri - 2)

Değerli Okurlarım Merhabalar,

Hatırlayacağınız gibi bir önceki makalemizde, Oyun Programcılığına girmek adına çarpışma tekniklerini incelemeye başlamış ve dörtgenlerin çarpışmalarını ele almıştık. Bugünkü makalemizde ise, dairesel şekillerin birbirleri ile olan çarpışmalarını incelemeye

çalışacağız. Dairesel şekillerin çarpışmasına verilebilecek en güzel örnek, kaynaklardan edindiğim bilgiye göre Bilardo oyunlarıdır. Burada gerçekten de mükemmel dairelerin birbirleriyle olan çarpışmaları söz konusudur. Şunuda hatırlatmakta fayda var. Şu an için teorilerimizi iki boyutlu uzayda inceliyoruz. Elbetteki işin için üç boyutlu cisimler girdiğinde kullanacağımız algoritmalar ve teknikler birazda olsa farklılık gösterecektir. Çünkü uzay boyutunda X ve Y koordinatlarına ek olarak Z koordinatları da işin içine girecektir. Bu da iki boyutlu bir sistemde Bilardo oyununun tasarlanmasının 3 boyutlu sistemdekine göre daha kolay olduğunu göstermektedir.

Dairesel nesnelerin çarpışmalarını belirlemek için eski dostumuz Pisagor Teoreminden faydalanacağız. Burada ana fikir, dairelerin merkezlerinin birbirlerine olan doğrusal uzaklıkları ile yarıçaplarının toplamlarının karşılaştırılmasıdır. Eğer, dairelerin merkezleri arası doğrusal uzaklık, dairelerin yarıçapları toplamından küçük ise, dairelerin üst üste geldiklerinden dolayısıyla çarpıştıklarından söz edebiliriz. Olayı aşağıdaki şekil ile ele almaya çalışalım.



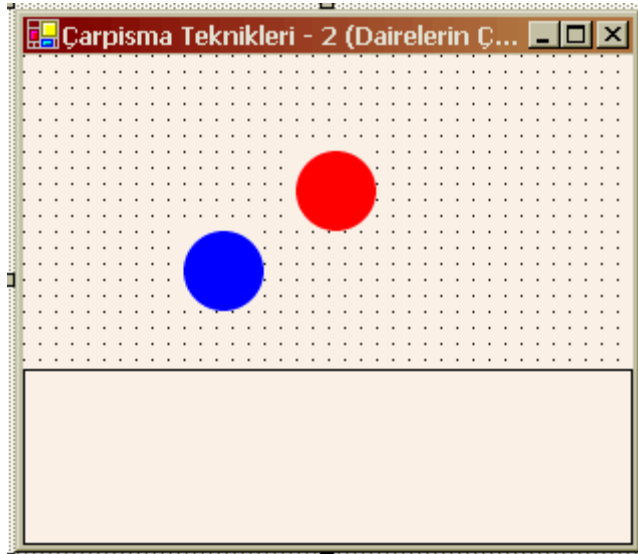
Şekil 1. Dairelerin Çarpışması.

Burada bizim için anahtar şekil, dairelerin merkezleri arasında oluşan BMA dik üçgenidir. Biz bu üçgen yardımıyla, A ve B noktaları arasındaki mesafeyi, başka bir deyişle BMA üçgeninin hipotenüsünü bulabiliriz. Bu bizim için en kilit noktadır. Sonrasında ise, bu mesafeyi R_A ve R_B yarıçaplarının toplamı ile karşılaştırmamız yeterli olacaktır. Dolayısıyla çarpışma formülümüz aşağıdaki gibi olmalıdır.

$$\begin{aligned}
 |X_A - X_B| &= |MA| \\
 |Y_A - Y_B| &= |MB| \\
 |AB| &= \sqrt{|MA|^2 + |MB|^2} \\
 |AB| < R_A + R_B &\Rightarrow \text{Çarpışma Var.}
 \end{aligned}$$

Şekil 2. Çarpışma Teorimiz.

Görüldüğü gibi yapmamız gereken, X ve Y koordinatları arasındaki mesafelerden yararlanarak, üçgenin hiptoneüsünü (yani iki daire merkezi arasındaki doğrusal uzaklığı) bulmak ve bulduğumuz değeri, yarıçapların toplamı ile karşılaştırmaktır. Şimdi dilerseniz, bu teoriyi C# ile geliştirilmiş bir windows uygulamasında nasıl simule edeceğimize bakalım. Geliştireceğimiz uygulama her zamanki gibi basit ve anlamsız olacak. Ancak amacımız, yukarıdaki teoremi uygulamak ve sonuçlarını görmek. Bu amaçla aşağıdaki ekran görüntüsüne ait bir windows uygulaması geliştirdim.



Şekil 3. Uygulama tasarımı.

Kodlarımız ise aşağıdaki gibi olacaktır.

```

/*global değişkenlerimizi tanımlıyoruz. */
float Top1_X,Top1_Y,Top2_X,Top2_Y,Top1_R,Top2_R;
double Hipotenus,YaricapToplam;

/*X,Y Koordinatlari ile yarıçaplar belirleniyor.*/

```

```

private void Hesapla()
{
    Top1_R=Top1.Width/2;
    Top1_X=Top1.Left+Top1_R;
    Top1_Y=Top1.Top+Top1_R;

    Top2_R=Top2.Width/2;
    Top2_X=Top2.Left+Top2_R;
    Top2_Y=Top2.Top+Top2_R;

    float Fark_X=Math.Abs(Top1_X-Top2_X);
    float Fark_Y=Math.Abs(Top1_Y-Top2_Y);

    Hipotenus=Math.Sqrt((Fark_X*Fark_X)+(Fark_Y*Fark_Y));

    YaricapToplam=Top1_R+Top2_R;
}

/* Ölçümleri ekrana yazdırmak için string değer döndüren bir metod hazırlıyoruz.*/
private string Olcumler()
{
    string olcum="Top 1 için X:"+Top1_X.ToString();
    olcum+=" Y:"+Top1_Y.ToString();
    olcum+=" R:"+Top1_R.ToString();
    olcum+=" | Top 2 için X:"+Top2_X.ToString();
    olcum+=" Y:"+Top2_Y.ToString();
    olcum+=" R:"+Top2_R.ToString();
    olcum+=" | Hipotenüs:"+Hipotenus.ToString();
    return olcum;
}

private void frmCollision2_Load(object sender, System.EventArgs e)
{
    Hesapla();
    lblOlcumler.Text=Olcumler();
}

/* Form üzerinde A,S,D,W tuşlarına basıldığında, Top2 isimli pictureBox' ın hareket etmesini sağlıyoruz.*/
private void frmCollision2_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)

```

```

{
    if(e.KeyChar==(Char)Keys.A)
    {
        Top2.Left-=1;
        Hesapla();
        lblOlculer.Text=Olculer();
    }
    if(e.KeyChar==(Char)Keys.D)
    {
        Top2.Left+=1;
        Hesapla();
        lblOlculer.Text=Olculer();
    }
    if(e.KeyChar==(Char)Keys.S)
    {
        Top2.Top+=1;
        Hesapla();
        lblOlculer.Text=Olculer();
    }
    if(e.KeyChar==(Char)Keys.W)
    {
        Top2.Top-=1;
        Hesapla();
        lblOlculer.Text=Olculer();
    }
    Kontrol();
}

/* Çarpışma kontrolümüzü yapıyoruz. */
private void Kontrol()
{
    if(Hipotenüs<YaricapToplam)
    {
        this.Text="";
        this.Text+=" |!!! ÇARPISMA VAR !!!| ";
    }
    else
        this.Text="";
}

```

Kodlarımız son derece açık. Dikkat etmemiz gereken noktalardan birisi, Top1 ve Top2 isimli nesnelerin X ve Y koordinatlarının bulunmasıdır. Burada Left ve Top özelliklerinin yanısıra daire merkezlerini tam olarak bulabilmek için, Width veya Height (daire olduklarından X veya Y'ye eklenecek mesafelerin Width veya Height ile hesaplanması farketmez) değerlerinden birisinin de göz önüne almamız gerekir. Yani aşağıdaki kodlarda olduğu gibi;

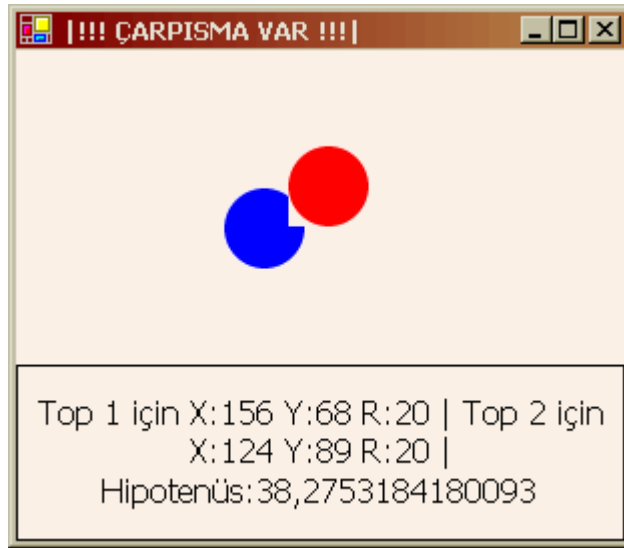
```
Top1_R=Top1.Width/2;
Top1_X=Top1.Left+Top1_R;
Top1_Y=Top1.Top+Top1_R;
```

```
Top2_R=Top2.Width/2;
Top2_X=Top2.Left+Top2_R;
Top2_Y=Top2.Top+Top2_R;
```

Bunun dışında Hipotenüs hesaplamasında elbetteki karekök almak için Math sınıfının Sqrt fonksiyonundan yararlanmaktayız.

```
Hipotenus=Math.Sqrt((Fark_X*Fark_X)+(Fark_Y*Fark_Y));
```

Uygulamamızı çalıştırdığımızda, Top2 isimli nesneyi herhangi bir yönden Top1 isimli nesne üstüne getirirsek çarpışmanın meydana geldiğini kolayca tespit edebiliriz.



Şekil 4. Çarpışmanın çalışma zamanında tespit edilmesi.

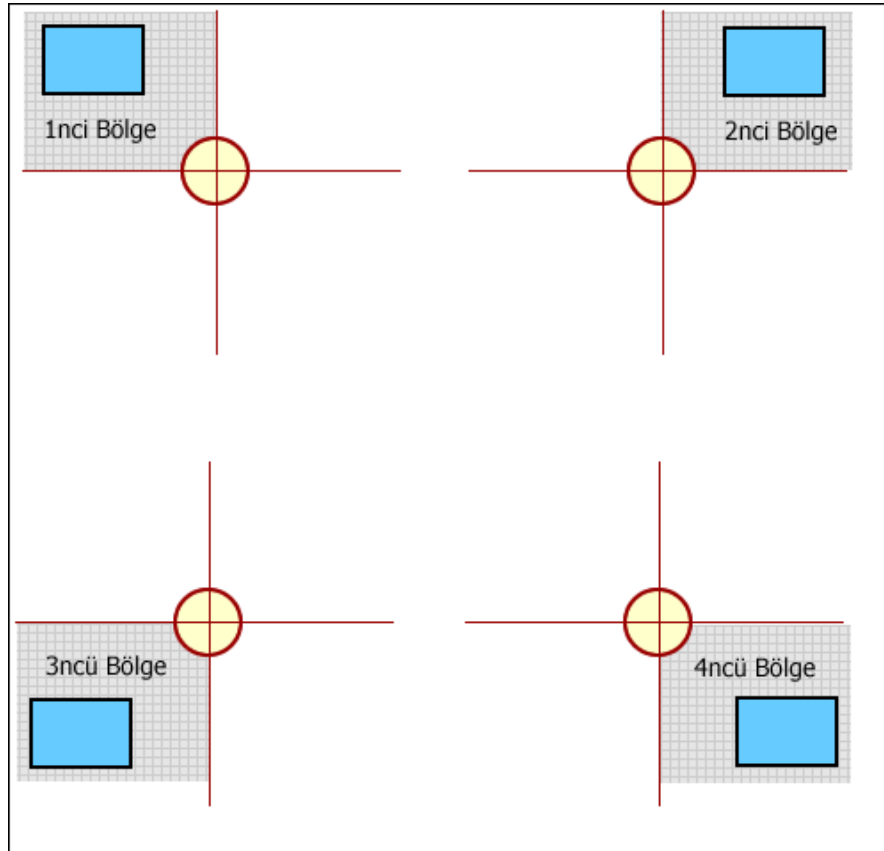
Çarpışmalar ile ilgili bir diğer önemli durumda, dörtgenler ile dairelerin çarpışmalarının nasıl tespit edilebileceğidir. Bu kez, daire merkez nesne olarak düşünülür ve dörtgenin daireye olan en yakın ve en uzak noktaları değerlendirilerek çarpışmanın olup olmadığına bakılır. Bu teoriyide bir sonraki makalemizde incelemeye çalışacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Oyun Programlamaya Giriş (Çarpışma Teknikleri - 3)

Değerli Okurlarım Merhabalar,

Geçtiğimiz hafta boyunca, Oyun Programcılığı ile ilgili olaraktan aldığım kitapları fırsat buldukça okumaya ve çalışmaya devam ettim. Konular o kadar heyecanlı ve sürükleyici ki araştırmak için zaman kavramı anlamsız hale geliyor. Öyleki, dün gece sabaha karşı saat 03:00 sularında kâğıt kalem ile boğuşuyor ve Çarpışma Tekniklerinden birisinin daha matematiksel modelinin C# ile nasıl uygulanabileceğini araştırıyordum. Sonuç olarak işe bir kaç saatlik uykuyla gitmek zorunda kaldım. Ancak buna rağmen tüm gün dinçtim. Çünkü, çarpışma tekniklerinden birisini daha öğrenmiştim. Sıra anlatmaya gelmişti. İşte bugünkü makalemizde 3ncü çarpışma tekniğini incelemeye çalışacağız.

Oyun programcılığında önemli bir yere sahip olan çarpışma tekniklerinde, bu makaleye gelinceye kadar iki ana konuyu inceleme fırsatı bulduk. İlk olarak iki dörtgensel nesnenin bir birleriyle olan çarpışmalarını inceledik. Daha sonraki makalemizde ise, eski dostumuz Pisagor' u anıp, iki dairesel nesnenin birbirleriyle olan çarpışmalarını araştırdık. Sırada bu iki durumun kombinasyonu var. Yani, bir dörtgen ile dairesel bir nesnenin birbirleriyle olan çarpışmalarının tespit edilmesi. Bu teknikte yine Pisagor teroeminden yararlanacağız. Ancak dikkat etmemiz gereken önemli koordinat noktaları var. Bu durumu daha iyi analiz etmek için aşağıdaki şekli göz önüne alalım.



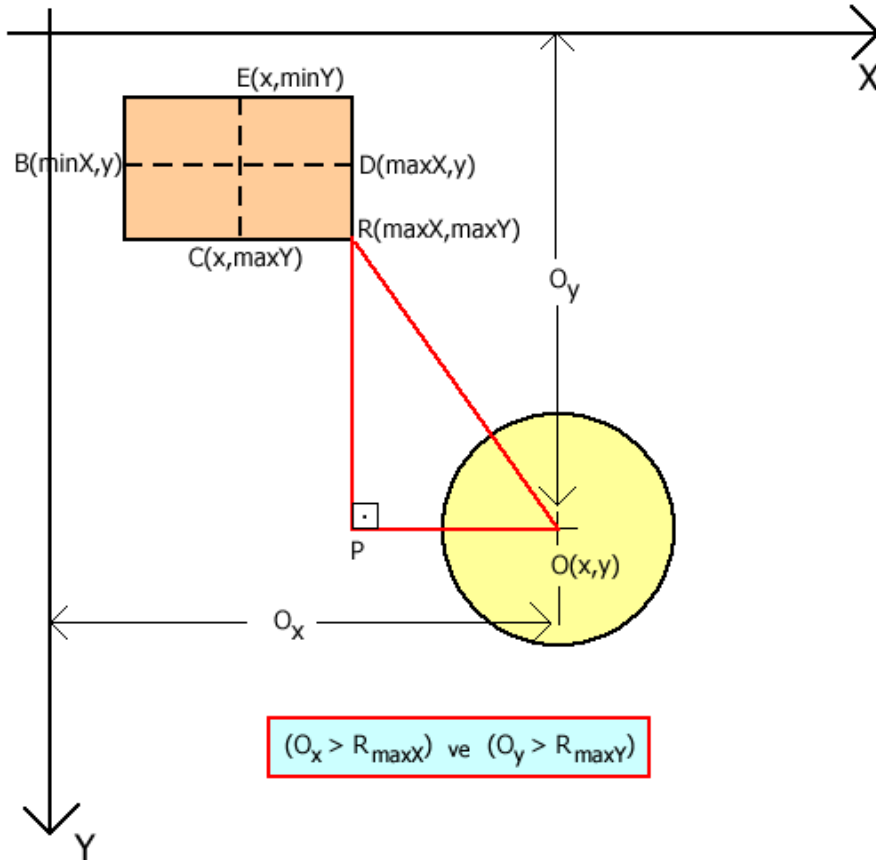
Şekil 1. Dört Bölge.

Bu teoride, dörtgenin dairesel nesneye göre olan konumlarını ele almamız gerekiyor. Bu da bizim, dörtgen nesnesine ait olan maksimum ve minimum sınır noktalarını bilmemiz gerektiğini göstermektedir. Buna göre, dörtgensel nesnenin köşe noktalarının x ve y koordinatları ele alınır. Kısaca, oluşturacağımız pisagor üçgeni için bu maksimum ve minimum x, y koordinatlarını bilmemiz gerekiyor. Çarpışma teoremine gelince;



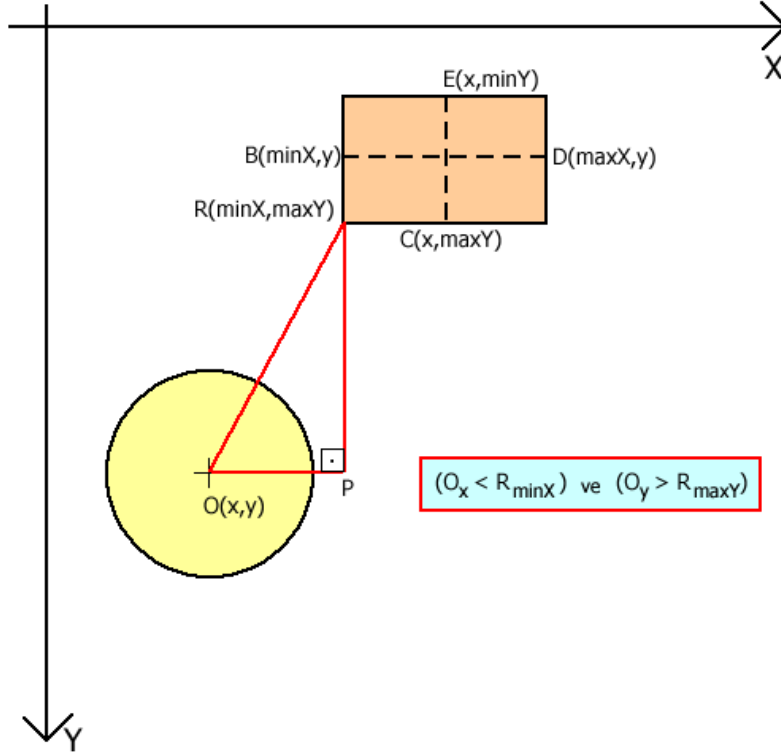
Dörtgenin dairesel nesneye en yakın olan köşe noktalarından yola çıkılarak oluşturulan dik üçgene ait hipotenüs değeri, dairesel nesnenin yarıçapından küçük ise çarpışma vardır.

Şimdi buradaki dört bölgeyi kısaca inceleyelim. Örnek olarak 4üncü bölgeyi aşağıdaki şekilde olduğu gibi ele alabiliriz. Esasen tüm bölgelerde, dörtgensel nesneye ait $\min X, \max X, \min Y$ ve $\max Y$ koordinatları büyük öneme sahiptir. Örneğin PRO üçgenini göz önüne aldığımızda, kenarların uzunluklarını bulabilmek için R noktasının koordinatlarından yararlanılmaktadır. R noktasına dikkat edecek olursanız, dörtgenin en uzak X ve en uzak Y değerlerine sahip olduğunu görürsünüz. Dolayısıyla biz, PO uzaklığını bulmak istiyorsak, Ox değerinden RmaxX değerini çıkartmamız gerekecektir. Aynı durum RP kenarının boyunu bulurken de geçerlidir. Bu durumda, Oy değerinden RmaxY değerini çıkartırız. Burada dörtgenin dördüncü bölgeye düştüğünü anlamak için dairenin x,y koordinatları ile R noktasının koordinatları karşılaştırılır. Burada dikkat etmemiz gereken husus, R noktasının koordinatlarını dörtgenin daireye olan konumuna göre değişecektir.

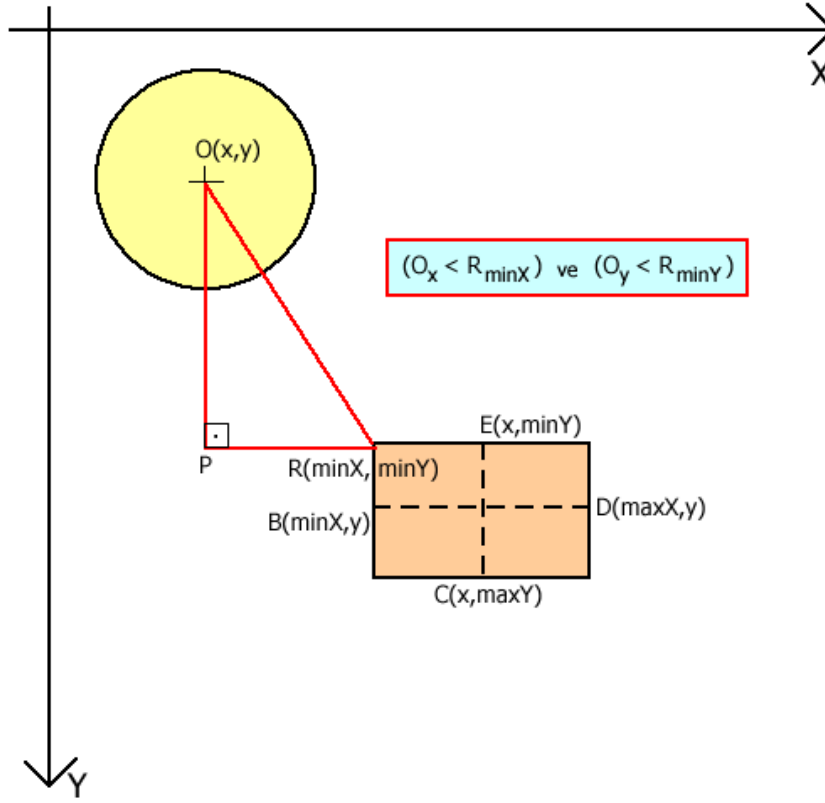


Şekil 2. Dördüncü bölge için durum.

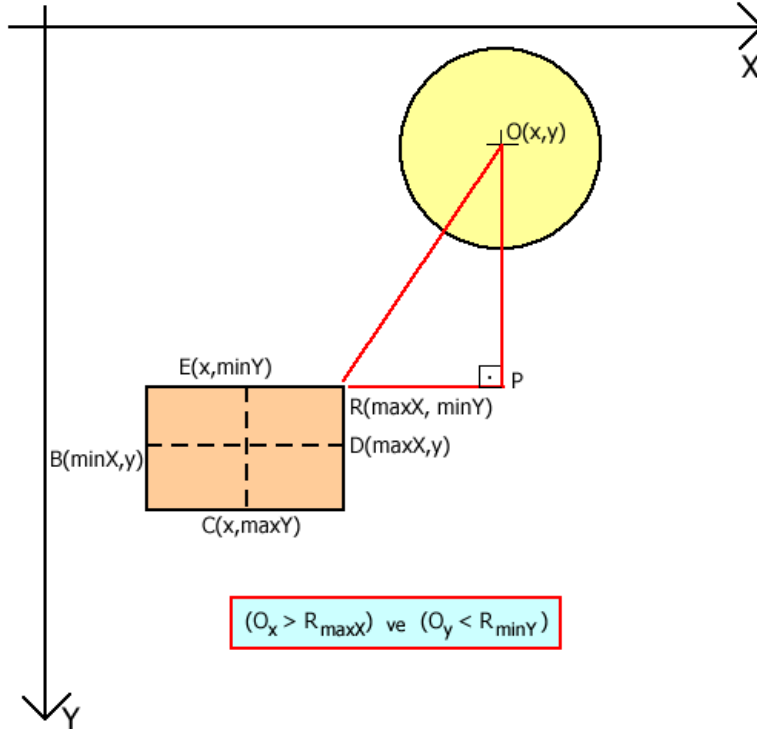
Bu anlattıklarım size biraz karmaşık geldiyse sıkı durun. Diğer bölgelerin grafiksel ifadeleride aşağıda yer almaktadır. Dikkat edecek olursanız, R noktasının koordinatları her bölge için farklıdır. Buda üçgene ait dik kenarlar hesaplanırken uygun R noktası koordinatlarını kullanmamız gerektiğini, başka bir deyişle programlama algoritmasında, söz konusu bölgeleri if koşulları ile tespit ederek geliştirmemiz gerektiğini gösterir.



Şekil 3. Üçüncü bölge için durum.

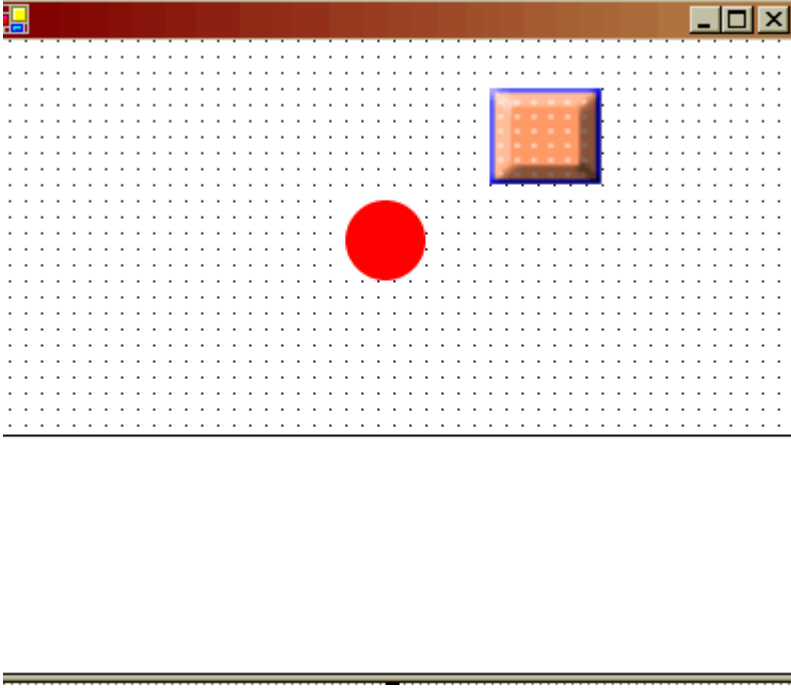


Şekil 4. Birinci bölge için durum.



Şekil 5. İkinci bölge için durum.

Şimdi sıra geldi bu teoremi C# kodları ile gerçekleştirmeye. Bu amaçla aşağıdaki ekran görüntüsüne sahip basit bir windows uygulaması geliştirdim. Her zamanki gibi işlemleri kolaylaştırmak amacıyla, dairesel nesneyi sabit tutup, dörtgensel nesneyi A,S,D,W tuşları ile hareket ettiriyorum. Bizim için önemli olan kısımlar, 4 durumdada geçerli olan üçgene ait dik kenarların FarkX ve FarkY şeklinde hesap edilmeleri ve buradan yola çıkılarak hipotenüs değerlerinin bulunması. Son olarak bu hipotenüs değerini, dairenin yarıçapı ile karşılaştırıp çarpışma olup olmadığını inceliyoruz.

*Şekil 6. Uygulama formu.*

Şimdi gelelim uygulama kodlarımıza.

```
/*global degiskenlerimizi tanimliyoruz. */
float Daire_X,Daire_Y,Dortgen_X,Dortgen_Y,Daire_R;
float Dortgen_MinX,Dortgen_MaxX,Dortgen_MinY,Dortgen_MaxY;
double Hipotenus,FarkX,FarkY;

/*Daire için X,Y koordinatlar, yarıçap Dörtgen için ise X,Y, minimum ve maksimum X,Y koordinatları hesaplanıyor.*/
private void Hesapla()
{
    Daire_R=Daire.Width/2;
    Daire_X=Daire.Left+Daire_R;
    Daire_Y=Daire.Top+Daire_R;

    Dortgen_X=Dortgen.Left+Dortgen.Width/2;
    Dortgen_Y=Dortgen.Top+Dortgen.Height/2;
    Dortgen_MinX=Dortgen.Left;
    Dortgen_MaxX=Dortgen.Left+Dortgen.Width;
    Dortgen_MinY=Dortgen.Top;
```

```
Dortgen_MaxY=Dortgen.Top+Dortgen.Height;
```

```
/*Burada temel olarak, dörtgenin hangi bölgelere denk düştüğüne bakarak dik üçgene ait FarkX ve FarkY değerlerini hesap ediyoruz.*/
```

```
if(Daire_Y<Dortgen_MinY)
{
    FarkY=Daire_Y-Dortgen_MinY;
}
if(Daire_Y>Dortgen_MaxY)
{
    FarkY=Daire_Y-Dortgen_MaxY;
}
if(Daire_X>Dortgen_MaxX)
{
    FarkX=Daire_X-Dortgen_MaxX;
}
if(Daire_X<Dortgen_MinX)
{
    FarkX=Daire_X-Dortgen_MinX;
}
/*Dik üçgene ait hipotenüsü buluyoruz.*/
Hipotenus=Math.Sqrt((FarkX*FarkX)+(FarkY*FarkY));
}
```

```
/* Ölçümleri ekrana yazdırmak için string deger döndüren bir metod hazirliyoruz.*/
private string Olcumler()
```

```
{
    string olcum="Daire için X:"+Daire_X.ToString();
    olcum+=" Y:"+Daire_Y.ToString();
    olcum+=" R:"+Daire_R.ToString();
    olcum+=" | Dortgen İçin X:"+Dortgen_X.ToString();
    olcum+=" Y:"+Dortgen_Y.ToString();
    olcum+=" | Dortgen İçin MinX:"+Dortgen_MinX;
    olcum+=" | Dortgen İçin MaxX:"+Dortgen_MaxX;
    olcum+=" | Dortgen İçin MinY:"+Dortgen_MinY;
    olcum+=" | Dortgen İçin MaxY:"+Dortgen_MaxY;
    olcum+=" | Hipotenüs:"+Hipotenus.ToString();
    return olcum;
}
```

```
/* Form üzerinde A,S,D,W tuslarına basıldığında, Dortgen isimli pictureBox' in hareket etmesini sagliyoruz.*/
private void frmCollision2_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
```

```
{
    if(e.KeyChar==(Char)Keys.A)
    {
        Dortgen.Left-=1;
        Hesapla();
        lblOlcumler.Text=Olcumler();
    }
    if(e.KeyChar==(Char)Keys.D)
    {
        Dortgen.Left+=1;
        Hesapla();
        lblOlcumler.Text=Olcumler();
    }
    if(e.KeyChar==(Char)Keys.S)
    {
        Dortgen.Top+=1;
        Hesapla();
    }
}
```

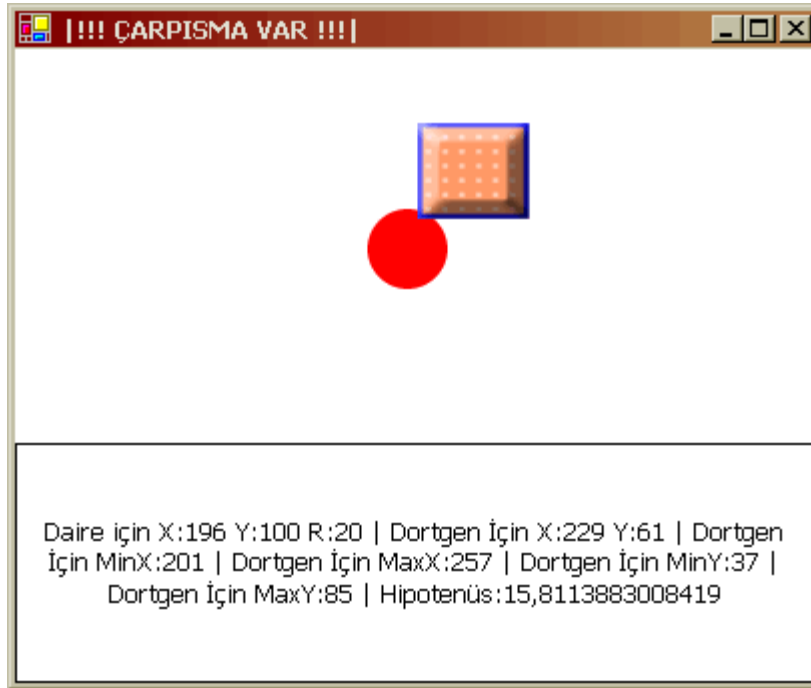
```

        lblOlcumler.Text=Olcumler();
    }
    if(e.KeyChar==(Char)Keys.W)
    {
        Dortgen.Top-=1;
        Hesapla();
        lblOlcumler.Text=Olcumler();
    }
    Kontrol();
}

/* Çarpisma kontrolümüzü yapıyoruz. */
private void Kontrol()
{
    if(Hipotenus<Daire_R)
    {
        this.Text="";
        this.Text+=" |!!! ÇARPISMA VAR !!!| ";
    }
    else
        this.Text="";
}

```

Kodlarımız son derece açık ve kolay. Uygulamamızı çalıştırdığımızda, çarpışma teorimizin başarılı bir şekilde çalıştığını görürüz.



Şekil 7. Çarpışma durumu.

Görüldüğü gibi artık oyun programlamada önemli noktalardan birisi olan çarpışma teorilerinde bayağı bir yol aldık. Ancak her zaman bu teknikleri kullanmayacağız. Örneğin zaman zaman, ekranın eşit karelere bölündüğünü (aynı kareli defterler gibi) ve nesnelerin bu kareler üzerindeki konumlarına göre çarpışıp çarpışmadıklarının belirlendiğini öğrendim. Önümüzdeki hafta büyük bir ihtimalle bu konuyu incelemeye çalışacağım. Kaynaklardan incelediğim kadarı ile bu karelere bölme tekniği tam anlamıyla PackMan

tarzı oyunlara yönelik geliştirilmiş bir model. İşin içine bu kez matris dizileri girecek. Bakalım başımıza neler gelecek. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Oyun Programlamaya Giriş (Matrisler Yardımıyla Çarpışma Kontrolü)

Değerli Okurlarım Merhabalar,

Hafta sonu evde bilgisayarım başında internette gezinirken, tarihi oyunların anlatıldığı bir site ile karşılaştım. Aslında zaten eski oyunları araştırıyordum. Amacım bu oyunlara, oyun oynamak isteyen bir çocuk gözü ile değil, onların yapılarını ve çekirdeklerini anlamaya çalışacak bir yazılımcı gözüyle bakabilmektir. Sonuçta, içimdeki çocuk ağır basıp bir kaç tanesini saatlerce oynadım. Aralarında en çok hoşuma gidenlerden birisi PackMan'ı di. Packman, doğrusal düzlemde 4 yöne hareket edebilen bir kahramandı. Yolda kendisini rastgele konumlardan gelerek yakalamaya çalışan böceklerden kaçıyor ve bulduğu meyveleri yiyerekte puanlar topluyordu. Tam oyunu bitirmeme az kalmıştıki hiç beklenmedik bir şekilde böceklerden birisi tarafından yendim. Aslında ekrana bir süre donuk gözler ile bakmıştım. Nitekim, oyunu oynarken aklıma geçen gün okuduğum Oyun Programlama kitabı gelmişti.

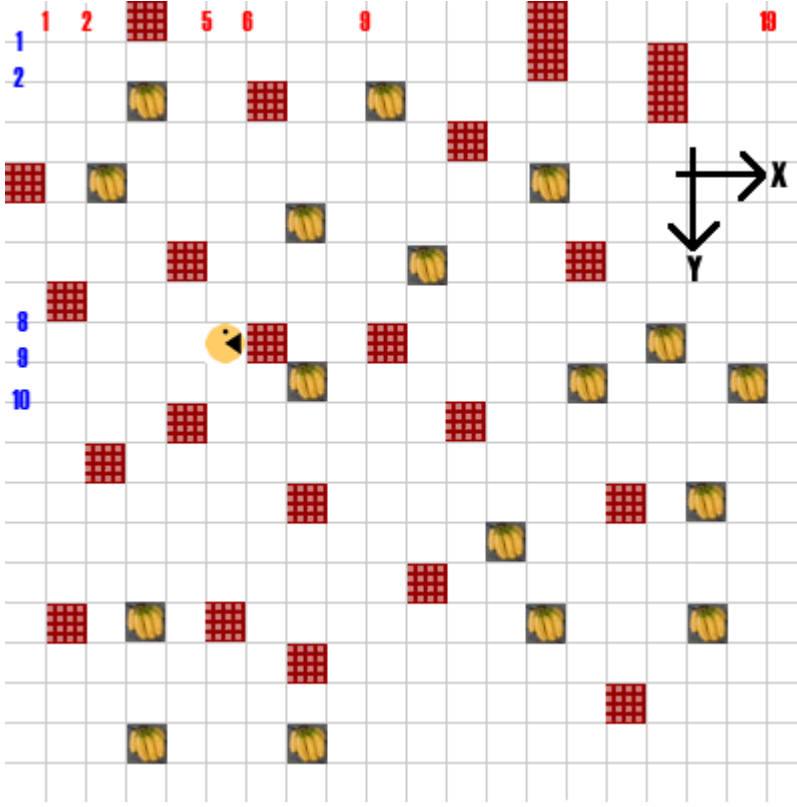
Kitabın bir bölümünde, ekranda yer alan aynı boyutlu nesnelerin çarpışmalarının kontrolünde iki boyutlu bir matris dizisinden faydalanılıyordu. O anda, çarpışma tekniklerinin farklı bir teoremini Packman'a benzeyecek bir oyun ile inceleyebileceğimi düşündüm. Her ne kadar amacım tümüyle bir oyunu yazmak olmasada en azından Packman'ı imi ekrandaki duvarların içinden geçirmeyecek şekilde hareket ettirmek istiyordum.

Teori gayet basitti. İlk olarak oyun sahasını, eşit boyutlu karelere bölecektim. Bu karelerin boyutları oyundaki nesnelerin çevresini saran hayali karelerinki ile aynı olacaktı. Böylece bir kare içinde her zaman tek bir oyun elemanı tam olarak sığmış bulunacaktı. Daha sonra ekrandaki elemanları iki boyutlu bir matris dizisi içinde bir şekilde konumlandıracaktım. Son olarak, kahramanın her hareketinde yöne bağlı olarak önceki veya sonraki kare alanlarını kontrol edecek ve orada bir Duvar nesnesi var ise çarpışma olduğunu belirterek o yöne olan hareketi kesecektim.



Teoremin kilit noktası, oyun alanındaki karelerde bulunan elemanları, iki boyutlu bir Matris dizisi içerisinde temsil edebilmektir.

İlk olarak aşağıdaki gibi bir oyun alanını düşündüm.



Şekil 1. Oyun Sahası.

Öncelikle, Duvar, Muz ve kahramanımız Packo' ya ait imajları tasvir ettim. Bunların her birisi 20 piksel X 20 Piksel boyutlarındaki bir karenin iç kenarlarına teğet olacak büyüklükteydiler. Daha sonra, oyun sahamı 20 Piksel X 20 Piksel' lik kareler ile doldurdum. Dolayısıyla artık elimde, 20' ye 20' lik bir Matris vardı. Bu Matris yardımıyla ekrandaki her bir elemanın konumunu bilebilirdim. Tek yapmam gereken Matrisin ilgili elemanına, orada duran nesneyi temsil edecek sayısal bir değer vermektir. Örneğin şu anki haliyle, Packo' nun Matris' deki konumunu aşağıdaki gibi ifade edebilirdim.

Matris[5,8]=1;

Bu durumda, Packo' nun sağa doğru olan hareketinde herhangi bir duvara çarpıp çarpmadığını kontrol etmek için, Y değerinin 1 fazlasına bakmak yeterli olacaktı.

Yani;

eğer Matris[5,8+1]=2 ise (ki Duvarlarında 2 sayısı ile ifade edebilirdim.)

Çarpışma var, Sağa gitme.

eğer Matris[5,8+1]=3 ise (ki Muzları 3 ile ifade edebilirdim.)

Sağa git, Matris[5,9] daki Muzu ekrandan sil, Puanı arttır.

eğer Matris[5,8+1]=0 ise (ne duvar ne de Muz var ise.)

Çarpışma yoktur. Yola devam et.

Bu kontrolü Packo' nun yapacağı doğrusal her hareket için uygulayabilirdim. Böylece ,



Hareket yönüne göre bir sonraki adımda yer alan elemanları Matris dizisi içinde bularak çarpışma kontrolünü gerçekleştirebilirdim.

Teoremi kafamda pekiştirdikten sonra, sıra bunu uygulamaya dökmeye gelmişti. Elbetteki bir Windows uygulaması için böyle bir teoremi araştırmaya çalışırken bir takım zorluklar ile karşılaşabilirdim. Örneğin, oyun başladığında Duvarların, Muzların ve Packonun rastgele ekrana konumlandırılması. Ekranda piksel bazında tutulan karesel alanların, Matris dizisi içerisinde nasıl indislenirebileceği. Öyle ya, 400 piksel' e 400 piksel' lik bir Form alanını, 400*400 elemanlı bir Matris dizisinde aynen uygulamak gereksiz yere hafıza tüketimine neden olurdu. Ya da, ekrandaki bir Muz' un üstünden geçildiğinde o resmin nasıl kaldırılacağı. Bu gibi pek çok sorunu önceden düşünmek ve uygulamayı ona göre planlamak gerekiyordu. Bu amaçla önce düşündüm ve sonra aşağıdaki kodları geliştirdim.

Konumlandırma işlemleri için kullandığım sınıf,

```
using System;

namespace Packo
{
    /* Bu sınıf yardımıyla oyun alanında kullandığımız nesnelerin X ve Y koordinatları ve duvar sayıları için rastgele değerler üretiriyoruz. Ekranımızı 20' ye 20'e lik karelere ayırdığımız için random sınıfının Next metodunu buna uygun şekilde çağırıyoruz. */
    public class Konumlandir
    {
        private int X,Y,duvarSayisi;
        System.Random r;

        public Konumlandir()
        {
            r=new Random();
        }

        /* X koordinatları için (herhangibir oyun elemanının Left özelliğinin değeri için) bir özellik tanımlıyoruz. Bu özellik Read-Only formatındadır. */
        public int YerlestirX
        {
            get
            {
                X=r.Next(1,20)*20; /* Oyun alanı 400 piksele 400 piksel boyutunda olduğu için, 1 ile 20 arasındaki rakami 20 kat sayısı ile çarpıyoruz.*/
            }
        }
    }
}
```



```

        return X;
    }
}

/* Y koordinatlari için (herhangibir oyun elemaninin Top özelliginin degeri için) bir özellik tanımlıyoruz. Bu özellik Read-Only formatındadır. */
public int YerlestirY
{
    get
    {
        Y=r.Next(1,20)*20;
        return Y;
    }
}

/* Ekrana 10 ile 30 arasında rastgele bir degerde Duvarlar koymak için bu özelligi kullanıyoruz. Bu özellik Read-Only formatındadır. */
public int DuvarSayisi
{
    get
    {
        duvarSayisi=r.Next(10,30);
        return duvarSayisi;
    }
}
}
}

```

Ana program;

```

Konumlandir k;

/* Ekrani 20*20 lik bir matris ile ele alacagiz. */
int[,] Matris=new int[20,20];

/* Matristeki her bir elemanin hangi oyun nesnesini (duvar,kahramanimiz packocuk ve Muz) temsil ettigini daha kolay kontrol edebilmek için sayisal degerleri bir enum sabiti ile anlamlandiriyoruz.*/
enum AlanSahibi
{
    Pakocuk=1,
    Duvar=2,

```

```

Muz=3
}

/* Ekrana duvar elemani, rastgele koordinatlara gelecek sekilde ekleniyor.*/
private void DuvarEkle()
{
    /*Bir PictureBox nesnesi tanimlaniyor.*/
    PictureBox pb=new PictureBox();
    /*Nesnemizin içerecegi resim yükleniyor. */
    pb.Image=System.Drawing.Image.FromFile("duvar.jpg");
    /*Nesnemizin ekrandaki yerlesimi için gerekli koordinat ayarlamalari yapiliyor.*/
    pb.Top=k.YerlestirY;
    pb.Left=k.YerlestirX;
    /* Duvarin ekrandaki piksel bazli koordinatlarini Matrisimizdeki elemanlar ile uyusturabilmek için 20 ile bölüyoruz.   Matrisin bu
    elemanina Duvar enum sabitinin degerinin veriyoruz.*/
    Matris[pb.Left/20,pb.Top/20]=(int)AlanSahibi.Duvar;
    /* Nesnemizin boyutlari belirleniyor.*/
    pb.Width=20;
    pb.Height=20;
    pb.SizeMode=PictureBoxSizeMode.StretchImage;
    /* Nesnemiz formumuzun Controls koleksiyonuna ekleniyor.*/
    this.Controls.Add(pb);
}

/* Ekrana bir Muz elemani, rastgele koordinatlara gelecek sekilde yerlestiriliyor.*/
private void MuzEkle()
{
    /*Muz nesnesinin üzerinden geçildiğinde onu ekrandan kaldırabilmek için kontrolün adını bilmem gerekiyor.Bunun için MuzEkle
    metodunun adını bulunduğu koordinata göre tanımlıyoruz.*/
    string ad;
    PictureBox pb=new PictureBox();
    pb.Image=System.Drawing.Image.FromFile("muz.jpg");
    pb.Top=k.YerlestirY;
    pb.Left=k.YerlestirX;
    Matris[pb.Left/20,pb.Top/20]=(int)AlanSahibi.Muz;
    ad="MUZ_" + Convert.ToString((pb.Left/20))+"_"+Convert.ToString((pb.Top/20));
    pb.Name=ad;
    pb.Width=20;
    pb.Height=20;
}

```

```

pb.SizeMode=PictureBoxSizeMode.StretchImage;
this.Controls.Add(pb);
}

/* Kahramanimiz Packo' nun ekrandaki konumu, Matris dizisindeki yeri ve saga veya sola bakacagi resmi belirleniyor. Ayrıca,
ekrana DuvarSayisi kadar Duvar ve Muz elemanlari ekleniyor.*/
private void Baslat()
{
    /* Rastegale X,Y ve duvar sayilari için kullandigimi Konumlandir sinifina ait bir nesne örneği oluşturuluyor.*/
    k=new Konumlandir();
    resPacko.Left=k.YerlestirX;
    resPacko.Top=k.YerlestirY;
    Matris[resPacko.Left/20,resPacko.Top/20]=(int)AlanSahibi.Pakocuk;
    /*Egere resPacko ekranin sol yarim küresinde ise, Sola bakan resmi gösteriliyor.*/
    if(resPacko.Left<200)
        resPacko.Image=System.Drawing.Image.FromFile("packoSol.jpg");
    /*Egere resPacko ekranin sag yarim küresinde ise, Saga bakan resmi gösteriliyor.*/
    if(resPacko.Left>200)
        resPacko.Image=System.Drawing.Image.FromFile("packoSag.jpg");
    resPacko.Visible=true;
    for(int i=1;i<k.DuvarSayisi;i++)
    {
        DuvarEkle();
        MuzEkle();
    }
}

int carpismaDurumu=0;

/* Eğer üstünden geçtiğimiz nesne Muz ise onu ekrandan kaldırıyoruz. Bunu yaparken Form üzerindeki kontrollerde gezinip,
kontrolün adını buluyor ve Remove metodunu çağırıyoruz.*/
private void MuzYokEt(int X,int Y)
{
    /*Önce Matrisimizin X,Y elemanının Muz olup olmadığına bakıyoruz.*/
    if(Matris[X,Y]==(int)AlanSahibi.Muz)
    {
        /*Eğer Muz ise dizinin bu elemanının değerini 0 yapıyoruz. Böylece Muz nesnemizi diziden çıkarmış oluyoruz.*/
        Matris[X,Y]=0;
        /* Daha sonra PictureBox' ımızın adını tedarik ediyoruz. */

```

```

string muzX=X.ToString();
string muzY=Y.ToString();
string KontrolAdi="MUZ_"+muzX+"_"+muzY;
/*Döngü ile, Form içindeki tüm kontroller arasında geziniyoruz.*/
for(int i=0;i<this.Controls.Count;i++)
{
    /*Eğer güncel kontrolün adı, bizim Muz nesnemizinki ile aynı ise, bu PictureBox' ı Formumuzdan çıkartıyoruz.*/
    if(this.Controls[i].Name.ToString()==KontrolAdi)
    {
        this.Controls.Remove(this.Controls[i]);
    }
}
}
}

/* Çarpisma kontrolünün yapıldığı metod. Bu metod Packo' nun X, Y koordinatları ile hareket ettiği yönü
(saga,sola,yukariya,asagiya) parametre olarak alıyor. Aldığı X,Y koordinatları ve yöne göre, bir sonraki kare alanında bir Duvar
nesnesi olup olmadigina bakıyor.*/
private void CarpismaKontrol(int X,int Y,char Yon)
{
    /*Eger sola hareket ediyorsak ve Matrisimizin [X indisinin 1 önceki elemanı,Y] AlanSahibi.Duvar enum sabitinin degerine esit ise
çarpisma vardır. */
    if(Yon=='A')
    {
        int alanSahibi=Matris[X-1,Y];
        if(alanSahibi==(int)AlanSahibi.Duvar)
            carpismaDurumu=1;
        else
            carpismaDurumu=0;
        MuzYokEt(X,Y);
    }
    /*Eger saga hareket ediyorsak ve Matrisimizin [X indisinin 1 sonraki elemanı,Y] AlanSahibi.Duvar enum sabitinin degerine esit
ise çarpisma vardır. */
    if(Yon=='D')
    {
        int alanSahibi=Matris[X+1,Y];
        if(alanSahibi==(int)AlanSahibi.Duvar)
            carpismaDurumu=1;
        else

```

```

        carpismaDurumu=0;
        MuzYokEt(X,Y);
    }
    /*Eger asagi hareket ediyorsak ve Matrisimizin [X,Y indisinin 1 sonraki elemanı] AlanSahibi.Duvar enum sabitinin degerine esit
ise çarpisma vardır. */
    if(Yon=='S')
    {
        int alanSahibi=Matris[X,Y+1];
        if(alanSahibi==(int)AlanSahibi.Duvar)
            carpismaDurumu=1;
        else
            carpismaDurumu=0;
        MuzYokEt(X,Y);
    }
    /*Eger yukari hareket ediyorsak ve Matrisimizin [X,Y indisinin 1 önceki elemanı] AlanSahibi.Duvar enum sabitinin degerine esit ise
çarpisma vardır. */
    if(Yon=='W')
    {
        int alanSahibi=Matris[X,Y-1];
        if(alanSahibi==(int)AlanSahibi.Duvar)
            carpismaDurumu=1;
        else
            carpismaDurumu=0;
        MuzYokEt(X,Y);
    }
}

/*Form üzerinde A (sol), S (asagi), D (saga), W (yukari) tuslarına basıldıkça çarpisma kontrolü yapiliyor. Eger çarpisma var ise,
belirtilen yöndeki dogrusal harekete (20 piksellik öteleme) izin verilmiyor. Aksi halde harekete devam ediliyor.*/
private void frmPacko_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
{
    int x=resPacko.Left/20;
    int y=resPacko.Top/20;

    if(e.KeyChar==(Char)Keys.A)
    {
        resPacko.Image=System.Drawing.Image.FromFile("packoSol.jpg");
        CarpismaKontrol(x,y,'A');
        if(carpismaDurumu==1)

```

```

        MessageBox.Show("DUVARA ÇARPTIN");
    else
        resPacko.Left-=20;
    }
    if(e.KeyChar==(Char)Keys.D)
    {
        resPacko.Image=System.Drawing.Image.FromFile("packoSag.jpg");
        CarpismaKontrol(x,y,'D');
        if(carpismaDurumu==1)
            MessageBox.Show("DUVARA ÇARPTIN");
        else
            resPacko.Left+=20;
    }
    if(e.KeyChar==(Char)Keys.S)
    {
        resPacko.Image=System.Drawing.Image.FromFile("packoAsagi.jpg");
        CarpismaKontrol(x,y,'S');
        if(carpismaDurumu==1)
            MessageBox.Show("DUVARA ÇARPTIN");
        else
            resPacko.Top+=20;
    }
    if(e.KeyChar==(Char)Keys.W)
    {
        resPacko.Image=System.Drawing.Image.FromFile("packoYukari.jpg");
        CarpismaKontrol(x,y,'W');
        if(carpismaDurumu==1)
            MessageBox.Show("DUVARA ÇARPTIN");
        else
            resPacko.Top-=20;
    }
}

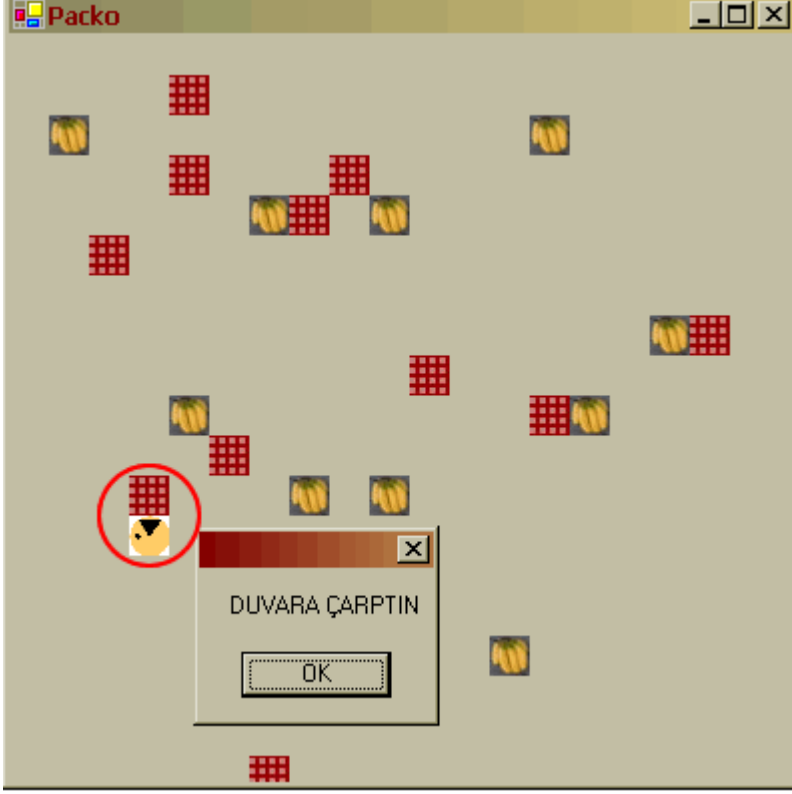
/*Oyunu kapatmak ve baslatmak kullandigimiz menüleri ContextMenu içerisinde kullaniyoruz.*/
private void menuItem3_Click(object sender, System.EventArgs e)
{
    Close();
}

```

```
private void menuItem1_Click(object sender, System.EventArgs e)
{
    Baslat();
}

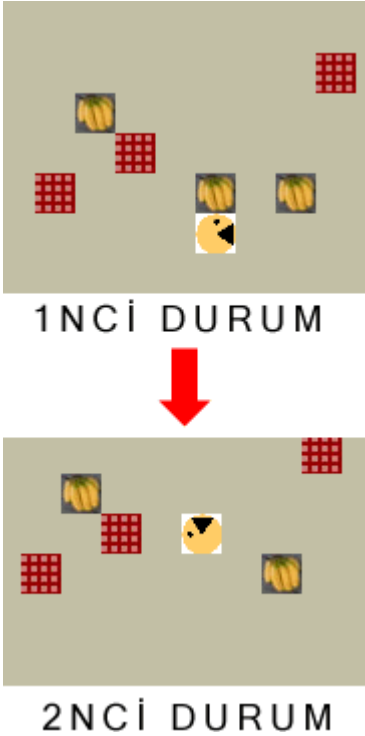
/* Kontrol amacıyla, Matris dizisinin degerlerini Text tabanlı bir dosyaya da yazabiliyoruz.*/
private void menuItem4_Click(object sender, System.EventArgs e)
{
    System.IO.FileStream fs=new
System.IO.FileStream("kontrol.txt",System.IO.FileMode.OpenOrCreate,System.IO.FileAccess.Write);
    System.IO.StreamWriter sw=new System.IO.StreamWriter(fs);
    for(int i=0;i<20;i++)
    {
        for(int j=0;j<20;j++)
        {
            sw.Write("{0,2},{1,2}={2,3}",i,j,Matris[i,j]);
        }
        sw.WriteLine();
    }
    sw.Flush();
    sw.Close();
    fs.Close();
}
```

Kodlar her ne kadar uzun görünsede programın tek yaptığı, Packo' yu duvarların içinden geçirmeden hareket ettirmek ve yolda gördüğü Muz' ları toplamasını sağlamak. Örneğin, uygulamayı çalıştırdığımızda bir duvara hangi yönden gelirsek gelelim, Matris dizimiz içinde Packo' dan sonraki elemanlar kontrol edilecek ve duvara çarpılıp çarpılmadığına bakılacaktır.



Şekil 2. Duvara Çarpış.

Diğer yandan, eğer Packo bir Muz üzerinden geçerse, bu Muz nesnesini temsil eden PictureBox Form üzerinden kaldırılacak, aynı zamanda Matris dizimizdeki ilgili Muz elemanın değeri de sıfırlanacaktır.



Şekil 3. Packo Muzları Yiyebiliyor.

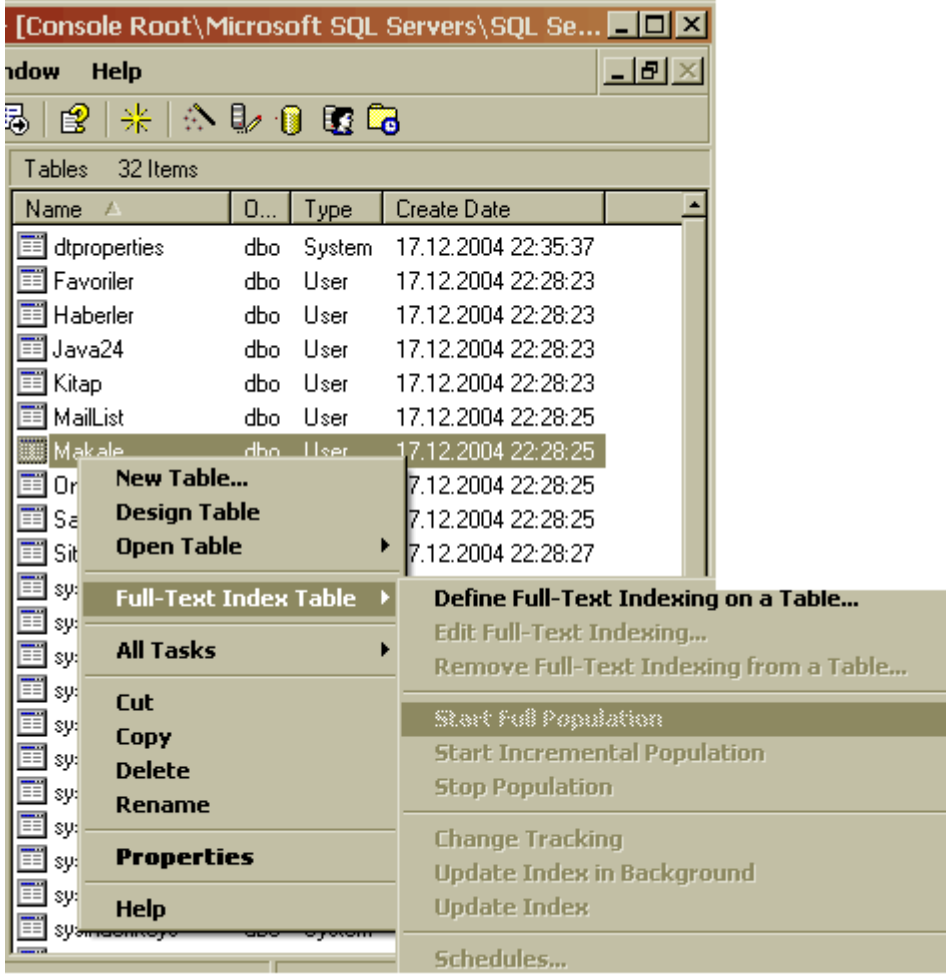
Görüldüğü gibi, Matris tekniği ile eşit karelere bölünmüş sahalardaki nesnelerin birbirleri ile olan çarpışmalarını kontrol edebilmek son derece basit. Bu minik program elbetteki başlangıç aşamasında. Örneğin, her muz yiyişinden sonra puanlama sistemi olması, duvarların seri olacak şekilde ekrana dizilmesi, Packo dışında hareket eden ve Packo'yu yemeye çalışan böceklerin farklı hareketleri vs... Bu kısımların geliştirilmesinde siz değerli okurlarıma bırakıyorum. Uygulamaya kaldığı yerden devam edebilirsiniz. Örnek uygulamayı indirmek için lütfen tıklayın. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Ado.Net ile Sql Server Full-Text Searching (Tüm Metinde Arama) Kullanımı

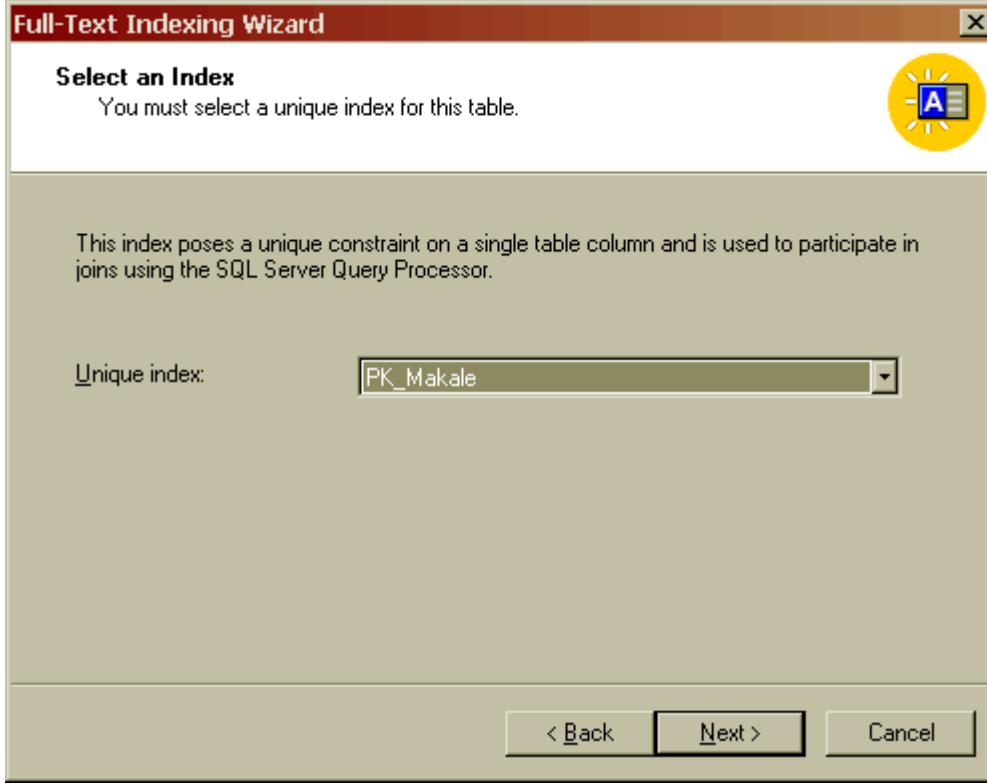
Değerli Okurlarım, Merhabalar.

Bu makalemizde, Sql sunucu üzerindeki tablolarda text tabanlı arama işlemlerinin, Full-Text Searching (Tüm Metinde Arama) servisi yardımıyla nasıl gerçekleştirildiğini incelemeye çalışacağız. Konunun pekişmesi açısından basit bir web uygulaması ile de bu hizmeti kullanıp text tabanlı arama işlemlerini irdelemeye çalışacağız. Özellikle dikkatinizi çekmiştir ki, internette yer alan pek çok arama motoru aradığımız kelimelerin geçtiği web sayfalarını bulup bize getirir. Çoğunlukla arama motoruna kayıtlı web sayfasının içeriğinde yapılan text tabanlı aramalar da Full-Text Searching (Tüm Metinde Arama) hizmetinden faydalanılır. Bu tip bir arama genellikle alanları içerisinde çok geniş text tabanlı içeriğe sahip olan tablolar üzerinde yapılmaktadır. Sql sunucusu, 7.0 versiyonundan itibaren bu hizmeti vermeye başlamıştır. Elbetteki arama işleminin gerçekleştirilebilmesi için Full-Text Searching (Tüm Metinde Arama) servisinin Sql sunucusunda yüklü olması gerekmektedir.

Dilerseniz hiç vakit kaybetmeden bir tablo için Full-Text Index' in nasıl oluşturulacağına kısaca bakalım. Öncelikle, Full-Text İndeksleme yapacağımız tablonun adına enterprise manager' dan sağ tıklayıyoruz ve aşağıdaki gibi **Define Full-Text Indexing On a Table**' ı seçiyoruz. Bu işleme başlamadan önce tablomuzda bir primary key alan olması gerektiğininde belirtelim. Nitekim indeksleme işlemlerinde bu alan, sonuçların döndürülmesinde anahtar alan olarak kullanılmaktadır.



Bu işlemin ardından karşımıza çıkan sihirbazdaki adımları birer birer işlemeye başlıyoruz. Az önce belirttiğim gibi bu ilk olarak tablomuzdaki primary alan için bir index seçiliyor.



Bu işlemin ardından, hangi kolonlarda arama yapılacağını belirtiyoruz. Full-Text aramalar çoğunlukla uzun text verilerin tutulduğu alanlarda kullanılır. Örneğin tablomuzdaki Icerik alanı makalelere ait Html içeriği barındırdan ntext veri tipindedir. Burada sadece karakter içerikli veya binary içerikli aramalara müsade eden alanların görüldüğünü söyleyelim. Yani primary key olan alanlar veya tarih formatındaki alanlar burada görünmez. Nitekim bu yapıdaki alanlarda Full-Text Indexing yapılması çok da anlamlı değildir. Artık Full-Text Searching (Tüm Metinde Arama) işlemimiz seçmiş olduğumuz alanlar üzerinde yapılabilecektir.

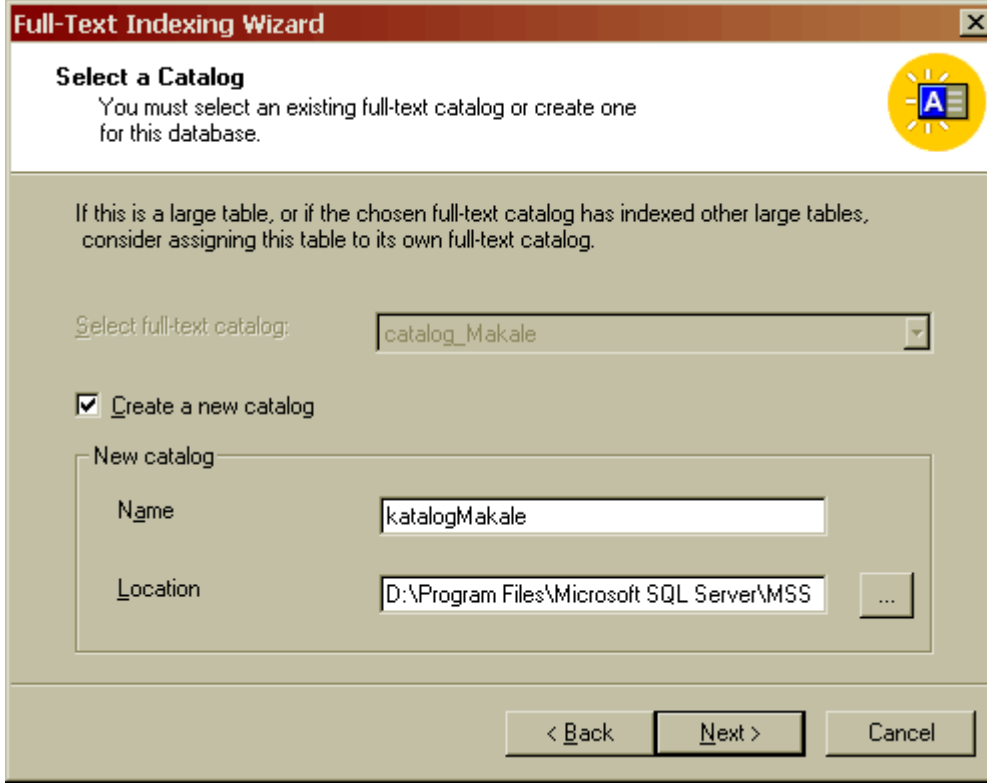
Full-Text Indexing Wizard

Select Table Columns
Select the character-based or image-based columns you want to be eligible for full-text queries.

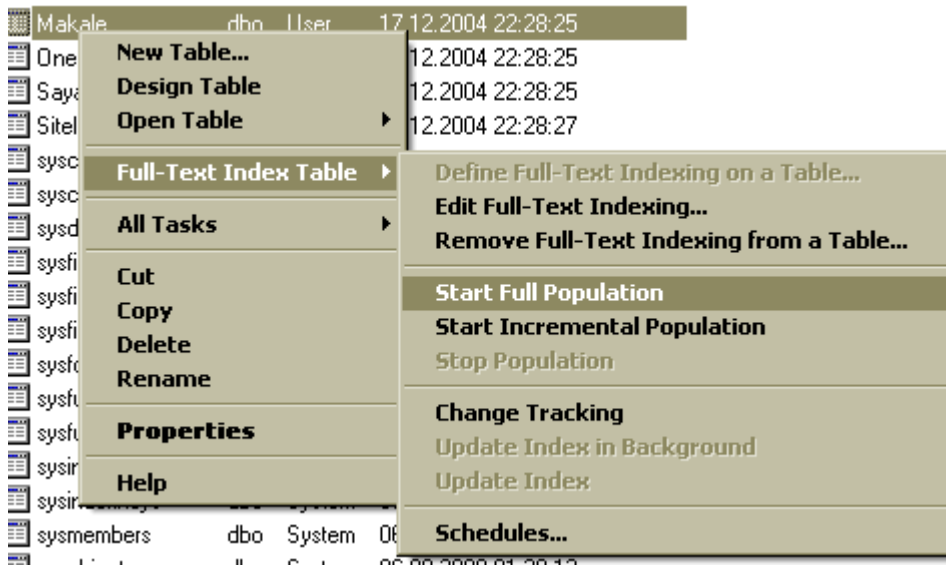
Available columns	Language for Word Breaker	Document type column
☒ Konu		
☒ İçerik		
☒ Kategori		

< Back Next > Cancel

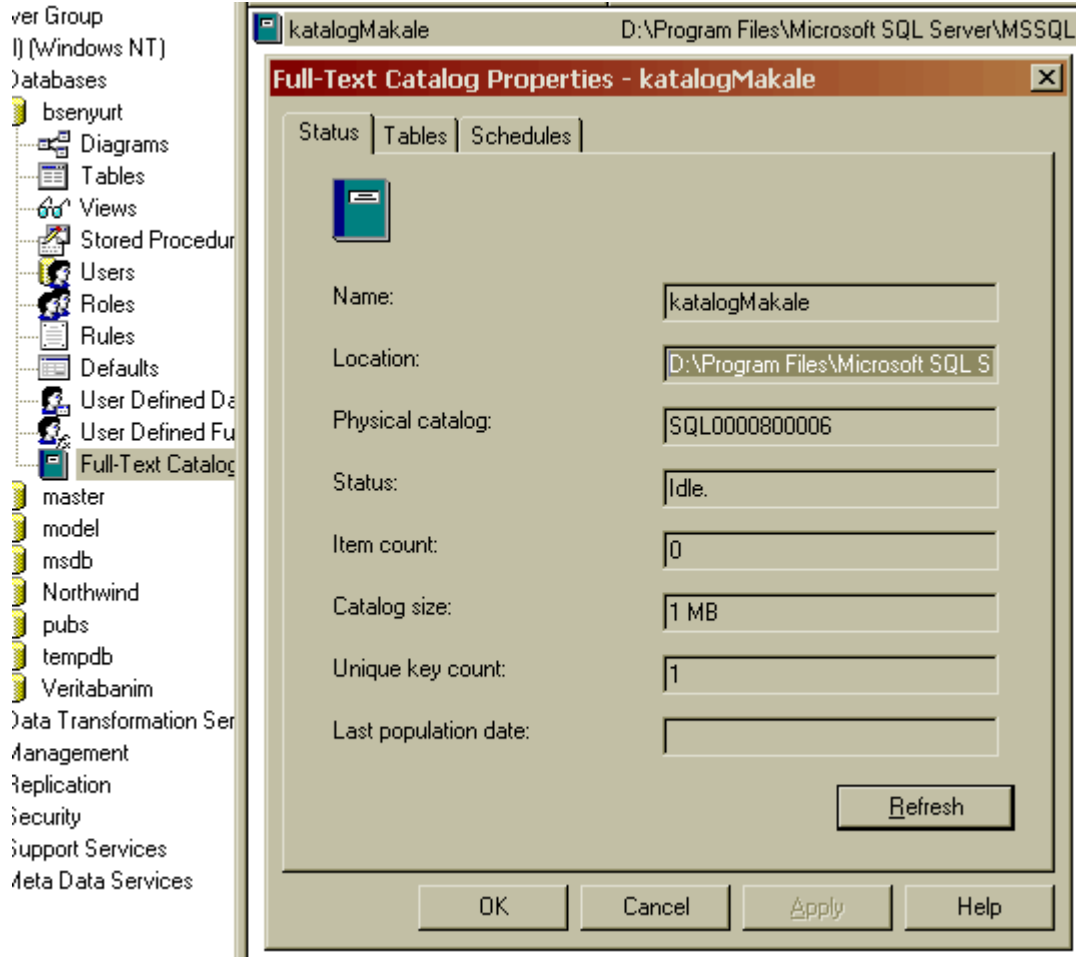
Sıradaki adımda ise, arama işlemi için gerekli katalog tanımlanır. Yani Full-Text Indexing aslında bir katalogda tutulmaktadır. Dolayısıyla bizim gerçekleştireceğimiz arama işlemleride bu kataloğu kullanacaktır. Kataloglar standart olarak d:\Program Files\Microsoft SQL Server\MSSQL\ftdata\ fiziki adresinde tutulur.



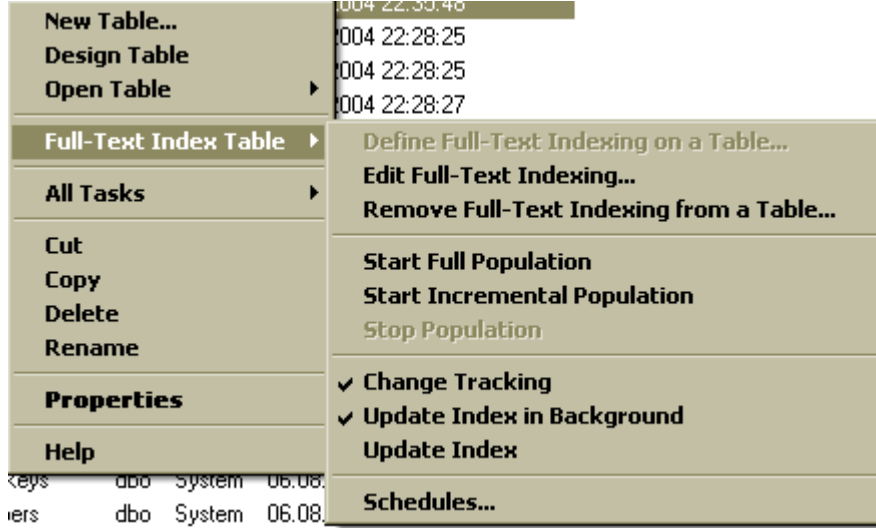
Daha sonraki adımlarda ise kataloğun belirli periyotlarda tekrardan doldurulmasını sağlayacak Schedule (Takvim) ayarlarında yapabiliriz. Bu ve izleyen adımları geçtikten sonra tek yapmamız gereken kataloğun doldurulması işlemidir. Bunun içinde, iki seçeneğimiz var. Birisi **Start Full Population**. Bu seçenek ile katalog baştan itibaren indekslenerek oluşturulur. Bir diğer seçeneğimiz ise **Start Incremental Population** dir. Bu seçenek sayesinde sadece tablonun eski hali ile yeni hali arasındaki farklar indeksleme işlemine katılarak katalog güncellenir. Bu daha önceden kataloglanmış indeksler için geçerli bir seçenektir. Şu an için biz kataloğu yeni oluşturduğumuzdan Start Full Population seçeneğini kullanacağız.



Bu işlemlerin ardından dikkat ederseniz, tablomuzun yer aldığı database içinde, **Full-Text Catalog** sekmesinde tanımladığımız kataloğun oluşturulduğunu görürsünüz.



Eğer tabloda oluşan güncellemelerden sonra katalog bilgisinin otomatik olarak yenilenmesini istiyorsak bu durumda **Change Tracking** özelliğın aktif hale getirmeliyiz. Bununla birlikte katalog güncelleme işleminin arka planda yapılmasını sağlamak için **Update Index in Background** seçeneğini aktif hale getirmeliyiz. Böylece, tablomuza yeni satırlar eklendiğinde, Full-Text Searching (Tüm Metinde Arama) işlemi için kullanılan katalog bilgilerinin otomatik olarak yenilenmesini sağlamış oluruz.



Artık tek yapmamız gereken Full-Text Searching (Tüm Metinde Arama) işlemini kullanmak. Bunun için T-SQL' de bir kaç komut var. Bunlardan ikisi, prototipleri aşağıdaki gibi olan **FREETEXT** ve **FREETEXTTABLE** anahtar sözcükleridir.

FREETEXT (Arama Yapılacak Alan , 'Aranacak Kelime')

FREETEXTTABLE (Tablo Adi , Arama Yapılacak Alan , 'Aranacak Kelime')

Her iki anahtar sözcükte, belli bir alan üzerinde Full-Text Searching (Tüm Metinde Arama) işlemini gerçekleştirmemizi sağlar. Ancak FreeTextTable sonuçları farklı bir tablo ile geri döndürür. Bu tabloda **Rank** ve **Key** isimli iki alan vardır. Key alanı, arama işleminin yapıldığı catalog oluşturulurken kullanılan primary key değerlerini alır. Rank alanı ise, aranan kelimelerin bulunduğu satırlar arasında bir derecelendirme yapılmasına imkan tanır.



Bu derecelendirme sayesinde aranan kelimenin daha çok geçtiği satırlardan, az geçtiği satırlara doğru (yada aranan kelimenin en uygun olarak eşleştirilebildiği alanlardan en az eşleştirildiği alanlara doğru) sıralanmış bir tablo görüntüsü elde etmemiz mümkündür.

İlk olarak Query Analyzer yardımıyla bu sorguların çalıştırılmasını inceleyelim.

```
SELECT ID, Konu, Tarih, Kategori FROM Makale
WHERE FREETEXT(Icerik, 'overload')
```

	ID	Konu	Tarih	Kategori
1	2	Stored Procedure Yardımıyla...	2003-11-12 00:00:00	Ado.Net
2	4	Overload Metodların Gücü	2003-11-13 00:00:00	C#
3	24	Reflection Sınıfı İle Tiple...	2003-12-22 00:00:00	C#
4	26	Kalıtım (Inheritance) Kavra...	2003-12-25 00:00:00	C#
5	34	Thread'leri Belli Süreler B...	2004-01-02 00:00:00	C#
6	47	İndeksleyiciler (Indexers)	2004-01-27 00:00:00	C#
7	92	Ado.Net 2.0 ve Bulk-Data Ko...	2004-09-17 00:00:00	Ado.Net

Burada görüldüğü gibi, FREETEXT anahtar sözcüğü, Icerik isimli alanda overload anahtar kelimesinin geçtiği satırların olduğu sonuç kümesini elde etmemizi sağlamıştır.

```
SELECT A.*, M.Konu, M.[ID] FROM Makale AS M INNER JOIN
FREETEXTTABLE(Makale, Icerik, 'overload') AS A ON M.[ID]=A.[KEY]
ORDER BY A.RANK DESC
```

	KEY	RANK	Konu	ID
1	4	504	Overload Metodların Gücü	4
2	2	330	Stored Procedure Yardımıyla...	2
3	92	274	Ado.Net 2.0 ve Bulk-Data Ko...	92
4	34	182	Thread'leri Belli Süreler B...	34
5	47	148	İndeksleyiciler (Indexers)	47
6	24	117	Reflection Sınıfı İle Tiple...	24
7	26	117	Kalıtım (Inheritance) Kavra...	26

FreeTextTable anahtar sözcüğünün kullanımı ise biraz daha karmaşıktır. Burada arama sonucu oluşan tabloya ait verileri aramanın yapıldığı tablo ile birleştirebilmek amacıyla **Join** tekniği kullanılmıştır. Join tekniğinde arama işleminde elde edilen tablonun **KEY** alanı ile Makale isimli tablonun primary key alanı olan ID alanları eşleştirilmiştir. Bu sorgu sonucunda **RANK** alanına göre tersten sıralı bir veri kümesi elde ederiz. Kümeyi Rank alanına göre tersten sıraladığımız takdirde, aranan kelimenin en uygun şartlarda eşleştirildiği satırlar en üste gelmiş olacaktır.

Şimdi Full-Text Searching (Tüm Metinde Arama) işlemini bir asp.net uygulamasında kullanalım. Bu uygulamada basit olarak bir arama işlemi sonucunda elde edilecek sonuçlar bir DataGrid kontrolünde gösterilecektir. Amacımız bir kelimeyi makalelerin içeriğinde aramak ve bulunan sonuçları kullanıcıya sunmak. Gerçekleştireceğimiz uygulamada aranan kelimenin girileceği TextBox'

ta oluşturulabilecek **Sql Injection**' ların önüne geçmek amacıyla bir **Stored Procedure** kullanacağız. Öncelikle sp' mizi aşağıdaki gibi oluşturalım.

```
CREATE PROCEDURE dbo.sp_AraBul
(
    @Aranan nvarchar(255)
)
AS

SELECT A.*,M.Konu,M.Tarih,M.[ID] FROM Makale AS M
INNER JOIN
FREETEXTTABLE(Makale,Icerik,@Aranan) AS A
ON M.[ID]=A.[KEY]
ORDER BY A.RANK DESC

RETURN
```

Sp' miz dışarıdan aranan kelimeyi parametre olarak alacak ve sonuçları çağırıldığı ortama döndürecek. Şimdide asp.net sayfamızı aşağıdaki şekilde oluşturalım.

Yalnız burada amacımız sadece aranan kelimenin geçtiği makaleleri bulmak değil. Aynı zamanda bulunan makalelerin olduğu sayfalara link vermek. Bu nedenle dataGrid kontrolümüzün içeriğini aşağıdaki gibi oluşturmamız gerekiyor.

```
<asp:DataGrid id="dgSonuclar" style="Z-INDEX: 105; LEFT: 48px; POSITION: absolute; TOP: 120px" runat="server"
AutoGenerateColumns="False">
<AlternatingItemStyle BackColor="#FFE0C0"></AlternatingItemStyle>
```

```

<ItemStyle ForeColor="Black" BackColor="#CCCCCC"></ItemStyle>
<Columns>
<asp:HyperLinkColumn DataNavigateUrlField="ID" DataNavigateUrlFormatString="MakaleGoster.aspx?ID={0}"
DataTextField="Konu" HeaderText="Makale Konusu"></asp:HyperLinkColumn>
<asp:BoundColumn DataField="Tarih" SortExpression="Tarih" ReadOnly="True" HeaderText="Yayın Tarihi"
DataFormatString="{0:dd-MM-yy}"></asp:BoundColumn>
</Columns>
</asp:DataGrid>

```

Şimdide uygulama kodlarımızı yazalım.

```

private SqlConnection con;
private SqlCommand cmd;
private SqlDataAdapter da;
private DataTable dt;

private void BaslangicAyarlari()
{
    con=new SqlConnection("data source=localhost;initial catalog=bsenyurt;integrated security=SSPI");
    cmd=new SqlCommand("sp_AraBul",con);
    cmd.CommandType=CommandType.StoredProcedure;
    cmd.Parameters.Add("@Aranan",SqlDbType.NVarChar,255);
    da=new SqlDataAdapter(cmd);
    dt=new DataTable();
}

private void Page_Load(object sender, System.EventArgs e)
{
    BaslangicAyarlari();
}

private void btnBul_Click(object sender, System.EventArgs e)
{
    cmd.Parameters["@Aranan"].Value=txtArananKelime.Text;
    try
    {
        if(con.State==ConnectionState.Closed)
            con.Open();
        da.Fill(dt);
        dgSonuclar.DataSource=dt;
        dgSonuclar.DataBind();
    }
}

```

```

}
catch(SqlException hata)
{
    Label1.Text=hata.Message.ToString();
}
finally
{
    if(con.State==ConnectionState.Open)
        con.Close();
}
}
}

```

Uygulamamızı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Aranan Kelime

overload

Sonuçlar

Makale Konusu	Yayın Tarihi
Overload Metodların Gücü	13-11-03
Stored Procedure Yardımıyla Tablodan Kayıt Silmek	12-11-03
Ado.Net 2.0 ve Bulk-Data Kopyalama Mekanizması	17-09-04
Thread'leri Belli Süreler Boyunca Uyutmak ve Yoketmek	02-01-04
İndeksleyiciler (Indexers)	27-01-04
Reflection Sınıfı İle Tiplerin Sırrı Ortaya Çıkıyor	22-12-03
Kalıtım (Inheritance) Kavramına Kısa Bir Bakış	25-12-03

Full-Text Searching (Tüm Metinde Arama) tekniğinde sadece tek bir kelime üzerinden arama yapmak zorunda değiliz. Örneğin "Matematik Mühendisi" kelimelerinin ardışık olarak geçtiği yerlerde kolayca bulabiliriz. Tabi böyle bir arama sonucunda "Matematik Mühendisi" kelimesi ile bire bir eşleşen metinlerin bulunduğu satırlar ile "mühendisi" kelimesinin olduğu ama "Matematik" kelimesinin olmadığı satırlarda elde edilcektir.

Diğer yandan içerik alanında hem overload kelimesi hemde interface kelimesi geçen makaleleri bulmak istediğimiz bir örnek ile karşılaşsak ne yaparız? İşte böyle bir durumda And, Or gibi mantıksal operatorlerin sunduğu imkanlardan faydalanmamız gerekecektir. Bunun için Full-Text Searching (Tüm Metinde Arama) işlemlerinde kullanabileceğimiz iki yeni sql anahtar sözcüğü vardır. Bunlar **CONTAINS** ve **CONTAINSTABLE** anahtar sözcükleridir. Bu ifadelerde **AND, OR, NEAR, *** gibi mantıksal

operatorler kullanılarak arama işlemleri daha detaylı bir şekilde gerçekleştirilebilir. Örneğin aşağıdaki sorgu, overload ve interface kelimelerinin bir arada geçtiği alanları arar.

```
SELECT A.*,M.Konu,M.[ID] FROM Makale AS M INNER JOIN
CONTAINSTABLE(Makale,Icerik,'overload AND interface') AS A ON M.[ID]=A.[KEY]
ORDER BY A.RANK DESC
```

Diğer yandan aynı aramayı overload veya interface kelimelerinden herhangi birinin geçtiği alanlar üzerinde de aşağıdaki sorgu ile gerçekleştirebiliriz. Tek yapmamız gereken aranan kelimeler arasına Or operatorunu koymak olacaktır.

```
SELECT A.*,M.Konu,M.[ID] FROM Makale AS M INNER JOIN
CONTAINSTABLE(Makale,Icerik,'overload OR interface') AS A ON M.[ID]=A.[KEY]
ORDER BY A.RANK DESC
```

	KEY	RANK	Konu	ID
1	4	103	Overload Metodların Gücü	4
2	38	34	Interface (Arayüz) Kullanım...	38
3	39	19	Arayüz (Interface), Sınıf (Cl...	39
4	2	14	Stored Procedure Yardımıyla...	2
5	40	10	Arayüzler'de is ve as Anaht...	40
6	92	9	Ado.Net 2.0 ve Bulk-Data Ko...	92
7	12	9	Struct (Yapı) Kavramı ve C...	12
8	22	5	HashTable Koleksiyon Sınıfı	22
9	24	4	Reflection Sınıfı İle Tiple...	24
10	26	4	Kalıtım (Inheritance) Kavra...	26
11	30	4	Boxing (Kutulamak) ve Unbox...	30
12	34	4	Thread'leri Belli Süreler B...	34
13	47	4	İndeksleyiciler (Indexers)	47
14	41	4	Bir Arayüz, Bir Sınıf ve Bi...	41

And ve Or operatörleri dışında kullanabileceğimiz bir diğer kullanışlı operatörde, * asteriks karakteridir. Örneğin,

```
SELECT A.*,M.Konu,M.[ID] FROM Makale AS M INNER JOIN
CONTAINSTABLE(Makale,Icerik,' "Datarela*" ') AS A ON M.[ID]=A.[KEY]
ORDER BY A.RANK DESC
```

sorgusu yardımıyla Datarela ile başlayan sözcükleri içeren alanların olduğu veri kümesini elde edebiliriz. CONTAINS ve CONTAINSTABLE aramalarında kullanabileceğimiz diğer operatörler için Sql Help' e bakabilirsiniz. Bu makalemizde Full-Text Searching (Tüm Metinde Arama) işlemlerinin nasıl yapıldığını incelemeye çalıştık. Umuyorum ki yararlı bir makale olmuştur. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Derinlemesine Session Kullanımı – 1

Değerli Okurlarım, Merhabalar.

Bu makalemizde, Asp.Net sunucularında durum yönetimi (state management) amacıyla kullanılan Session nesnesini detaylı bir şekilde incelemeye başlayacağız. Bildiğiniz gibi, web anatomisinde durum yönetimi başlı başına bir terminolojidir. Web sitelerinin yer aldığı web sunucularının istemci makinelerde durum yönetme yeteneğine sahip olamamaları sonucu, belli bir kullanıcıya ait bilgilerin sayfalar arasında korunması veya taşınabilmesi için değişik teknikler geliştirilmiştir. Session kullanımı bu tekniklerden birisidir. Öncelikle Session kavramının ne olduğunu ve ne için kullanıldığını kavramaya çalışalım.

Bir kullanıcı, bir web sunucusundan herhangi bir sayfa talep ettiğinde, web sunucusu bu kullanıcı için kendi sistemi üzerinde veya yardımcı bir sunucuda (web farm' lar göz önüne alındığında) bir oturum başlatır. İşte Session nesneleri yardımıyla kullanıcıya ait olan bu oturum boyunca sayfalar arasında bilgi taşıma işlemini gerçekleştirebiliriz. Session kullanımına verilebilecek en güzel ve klasik örnek, kullanıcılarının alış veriş sepetleri ile birlikte site içindeki sayfalar boyunca gezinebilmeleridir. Burada temel fikir kullanıcının sahip olduğu sepete ait bilgileri site içerisinde yer alan sayfalar boyunca taşıyabilmektir. Biz bu amaçla Session nesnelerini kullanabilir ve kullanıcının oturumu sonlanıncaya kadar sayfalar arasında veri taşıyabiliriz. Elbette başlayan bir oturum mutlaka bir şekilde sonlandırılır. Bir oturum temel olarak aşağıdaki nedenlerden dolayı sonlanabilir?

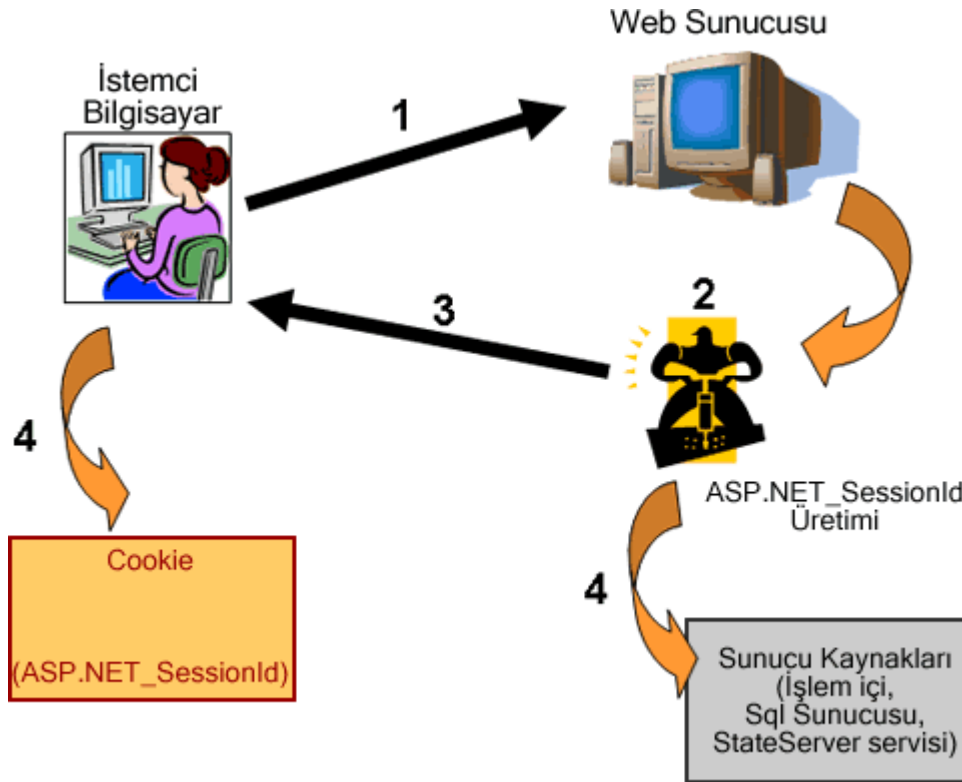
Oturum Sonlanmasına Neden Olacak Etmenler	
Oturumun zaman aşımına uğraması. (Timeout)	Varsayılan olarak sunucu üzerinde açılan her bir oturum 20 dakikalık zaman aşımı süresine sahiptir. Sitemizi ziyaret eden kullanıcıların Session verileri ile ne kadar süre durabilecekleri göz önüne alınarak bu süre daha da düşürülebilir veya arttırılabilir. Sürede yapılan ayarlamalar kesinlikle sunucuya ait bellek tüketimini etkileyecektir.
Kullanıcının tarayıcı penceresini kapatması veya sayfaya yeni bir tarayıcı penceresi açarak ulaşması.	Bir web sunucusu kullanıcının tarayıcısını kapatıp kapatmadığını anlayamaz. Onun yerine barındırdığı sayfalara yapılan çağrılarını tespit eder. Dolayısıyla bir kullanıcı açık bir oturuma sahip iken tarayıcısını kapattığında veya yeni bir tarayıcı penceresi açtığında aynı sayfaya yeni bir oturum içinde bağlanır. Bunun sebebi sunucunun ürettiği ASP.NET_SessionId' lerin her bir istekte yeniden oluşturulmasıdır.
Oturum kapatılmaya zorlandığında.	Session nesnesine Abandon metodu uygulanarak oturumlar iptal edilebilir.
Sunucu yeniden başlatıldığında.	Bir sunucunun yeniden başlatılması eğer SessionId değerleri işlem içinde (yani asp.net work processor' un açtığı iş parçalarında) tutuluyorsa otomatik olarak kaybedilecektir. Ancak SessionId lerin bir Sql sunucusunda database' de veya StateServer isimli windows servisinde tutulması sağlanarak bu kayıpların önüne geçilebilir.

Gelelim Session nesnelerinin sunucu tarafından doğru istemciler ile nasıl eşleştirilebildiğine. Web sunucusundan bir sayfa talep edildiğinde **(1)** Asp.Net Work Processor bu istemci için otomatik olarak benzersiz bir tanımlama (Unique Identity) değeri üretir **(2)**. Bu 120 bitlik bir sayıdır ve özel bir algoritma yardımıyla sunucu üzerinde oluşturulur. Daha sonra bu talep karşılığında sunucu, oluşturduğu ASP.NET_SessionId değerini istemci bilgisayara gönderir **(3)**. Bu ASP.NET_SessionId değerinin istemci üzerinde tutulduğu varsayılan olarak bir çerez (Cookie) vasıtasıyla sağlanır **(4)**. Aynı değer Sunucu üzerinde de yer almaktadır **(4)**. Bu noktadan itibaren istemci, kendisine ait oturumu sunucu üzerinde bir şekilde sonlanıncaya kadar, site içerisinde bu ASP.NET_SessionId değeri ile tanınır. Dolayısıyla artık sunucu, oturum sahibinin kim olduğunu bilmektedir.



Tanıtma; Sunucular oturum sahiplerini tanıyabilmek için, özel bir algoritma ile oluşturdukları 120 bitlik benzersiz bir tanımlayıcı değeri kullanırlar. **ASP.NET_SessionId**.

Ancak burada istisnai bir durum vardır. Her istemci cookie' leri desteklemez. Bu durumda yapılacak ufak bir ayarlama ile ASP.NET_SessionId değerinin url' ye eklenmesi sağlanır. Bunu ilerleyen safhalarda inceleyeceğiz. Aşağıdaki şekil basit olarak bir SessionId değerinin nasıl oluşturulduğunu betimlemektedir. Bu varsayılan, yani Cookie desteği veren tarayıcılar için geçerli olan senaryodur.



Session nesnesine atanan veriler web sunucusu üzerindeki bellekte tutulmaktadır. Bu bilgiler asla istemci bilgisayara gönderilmezler. Bu da güvenlik açısından önemli bir avantajdır.



Güvenlik; Session nesnelere ait veriler Sunucu üzerindeki bellek alanlarında tutulur ve asla istemcilere gönderilmez.

Burada dikkat etmemiz gereken önemli bir husus vardır. Session bilgilerinin sunucu belleğinde (varsayılan olarak budur) tutulması. Bir Session nesnesinin en büyük özelliklerinden birisi serileştirilebilir (Serializable) nesneleri taşıyabilme özelliğine sahip olmasıdır. Yani, bir Session nesnesi yardımıyla kullanıcının oturumu boyunca bir DataSet nesnesini sayfalar arasında dolaştırabiliriz. Peki burada dikkat etmemiz gereken nokta nedir? Sorun DataSet gibi bir nesnenin tek başına çalıştığı bir bilgisayar için bile fazla bellek alanı harcamasıdır. Yani, sunucuya bağlanan her bir kullanıcı için oturum süresi boyunca DataSet nesne örneklerinin taşınması sunucu kaynaklarını ciddi ölçüde azaltır. Bellekte meydana gelen bu azalma elbetteki web sunucusunun performansını olumsuz yönde etkileyecektir.



Performans; Bir web uygulamasında Session nesnelerinin DataSet nesne örnekleri taşıdığını varsayarsak, N sayıda kullanıcının bağlanması, sunucu belleğinde N sayıda DataSet nesne örneğinin oluşturulması anlamına gelmektedir.

Peki çözüm olarak ne üretebiliriz? İlk olarak önbellekleme tekniklerini kullanabiliriz. Ya da Serileştirilebilir nesneleri örneklendiren sınıfları kendimiz yazarız. Tabi böyle bir durumda yazacağımız sınıflarında serileştirilebilir olması bir başka deyişle serileştirmeye izin veren nitelikleri (Attribute) kullanması gerekecektir. Bu konuyu ilerleyen makalelerimizde incelemeye çalışacağız. Elbetteki bir Session sadece serileştirilebilir nesneleri değil her tipte .NET nesne örneğini saklayabilir.

Bu kısa açıklamalardan sonra dilerseniz, Session nesnelerinin nasıl oluşturulduğuna ve kullanıldığına kod kısmından basit bir örnek ile bakmaya çalışalım. Aşağıdaki uygulamada, default.aspx, sayfa2.aspx ve sayfa3.aspx isimli 3 adet web sayfamız var. Biz default.aspx içerisinde kullanıcıdan isim ve soyisim bilgisini alıyor ve bunu Session nesnemizde tutuyoruz. Daha sonra sayfa2.aspx ve sayfa3.aspx içerisinden bu session değerlerini okuyup ekrana yazdırıyoruz. Default.aspx ilk açıldığında, üretilen ASP.NET_SessionId değerini de Session nesnesinin SessionId özelliği ile elde ediyoruz. Örnek uygulamayı [bu linkten](#) indirebilirsiniz.

```
//default.aspx
private void btnEkle_Click(object sender, System.EventArgs e)
{
    Session["Kimlik"] = txtBilgi.Text; //Session' a değer atıyoruz.
}

//sayfa2.aspx
private void Page_Load(object sender, System.EventArgs e)
{
    lblAdSoyad.Text = Session["Kimlik"].ToString(); //Session değerini okuyoruz.
}

//sayfa3.aspx
```

```
private void Page_Load(object sender, System.EventArgs e)
{
    lblAdSoyad.Text=Session["Kimlik"].ToString(); //Session değerini okuyoruz.
}
```



Yukarıdaki örnek Session kullanımının en basit halidir.

Oturum yönetiminde işimize yarayacak iki olay vardır. Session_Start ve Session_End olayları. Bir istemci bir web sunucusunda bir oturum başlattığında uygulamaya ait global.asax dosyasında yer alan Session_Start olayı çalışır. Oturum herhangi bir neden ile sonlandığında ise yine global.asax dosyasındaki Session_End olayı çalışır. Bu olaylar yardımıyla, uygulamanızda oturum halinde bulunan kullanıcı, sayısını bir başka deyişle sayfadaki ziyaretçi sayısını öğrenebilirsiniz. Örneğin az önceki uygulamamızı ele alalım ve Global.asax dosyasında aşağıdaki değişiklikleri yapalım. Uygulamamızdaki default.aspx sayfasında da aktif ziyaretçi sayısını göstermek için bir label kullanacağız.

```
protected void Session_Start(Object sender, EventArgs e)
{
    /*Uygulama seviyesinde bir değişken kullanmak istediğimizden Application nesnesini kullandık. Eğer ilk kez oturum açılıyor ise varsayılan değeri 1 olarak ayarlıyoruz. İlk kez açılmıyorsa var olan değeri 1 arttırıyoruz.* /
    if((int)Application["ToplamZiyaretci"]==0)
        Application["ToplamZiyaretci"]=1;
    else
    {
        int deger=(int)Application["ToplamZiyaretci"];
        deger+=1;
        Application["ToplamZiyaretci"]=deger;
    }
}

protected void Session_End(Object sender, EventArgs e)
{
    /* Oturum kapatıldığında devereğe giren bu olayda, uygulama seviyesindeki Application nesnesinde saklanan değerini 1 azaltıyoruz. Böylece online ziyaretçi sayısını 1 azaltmış oluyoruz.* /
    int deger=(int)Application["ToplamZiyaretci"];
    deger-=1;
    Application["ToplamZiyaretci"]=deger;
}
```

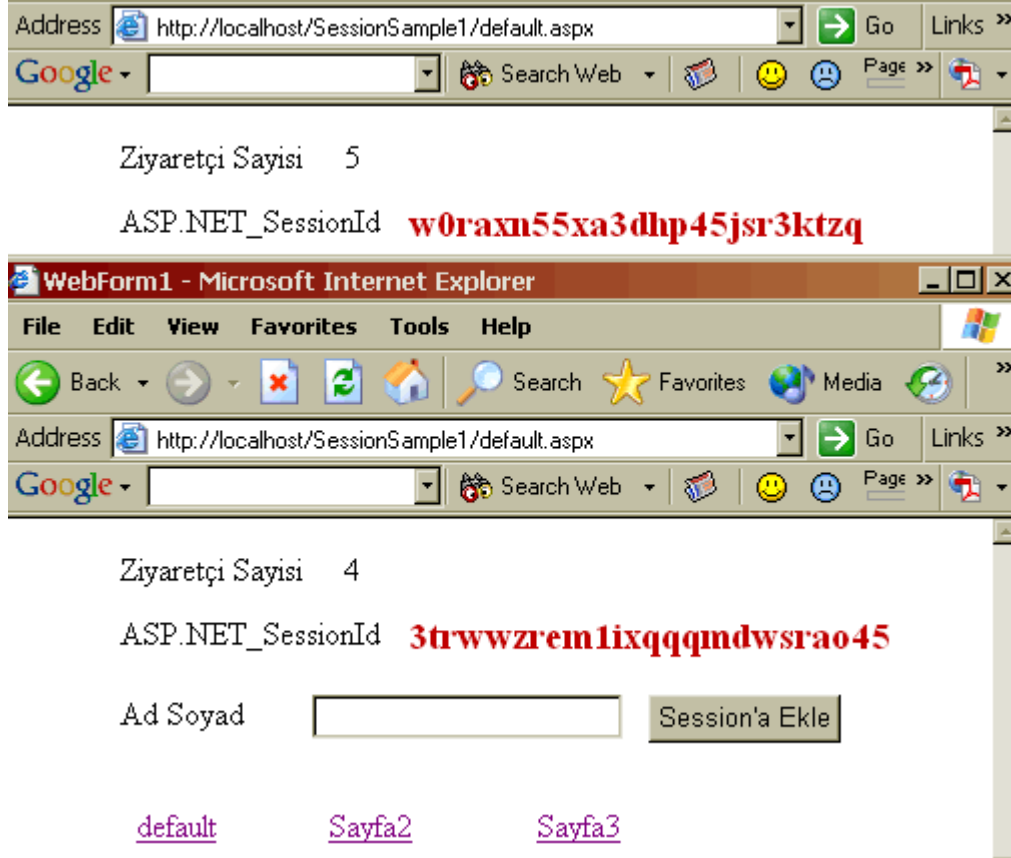
Kodda dikkat ederseniz Application nesnesinin değerini deger isimli değişkene alırken (int) tür dönüştürme operatörünü kullandık. Bunun sebebi, Application nesnesinde Session nesnesi gibi verileri object tipinde saklıyor olmasıdır. Dolayısıyla Session veya Application nesnelerinin değerlerini okurken uygun tür dönüşümlerini açıkça yapmamız gerekiyor. Artık tek yapmamız gereken Application nesnesinin değerini okumak ve label kontrolümüze yazdırmak.

```
private void Page_Load(object sender, System.EventArgs e)
{
    Session["Kimlik"]=txtBilgi.Text;
    lblZiyaretciSayisi.Text=Application["ToplamZiyaretci"].ToString();
}
```

Bundan sonra uygulamamıza her yeni oturum açışımızda, Session_Start devreye girecek ve kullanıcı sayısını otomatik olarak 1 arttıracaktır. Session'lar sonlandığında ise 1 eksileceklerdir. Bunu daha iyi kavrayabilmek için oturumların timeout süresini 1 dakikaya indirelim. Session nesnesinin Timeout özelliğine 1 değerini atayarak bunu sağlayabiliriz.

```
Session.Timeout=1;
```

Şimdi peş peşe tarayıcı pencereleri açıp uygulamamızın olduğu default.aspx sayfasını çağıralım.



Burada da görüldüğü gibi, Session_Start olayı başarılı bir şekilde çalışarak Application nesnesinin ToplamZiyaretci elemanının değerini sürekli olarak 1 arttırmıştır. Lakin biz Session Timeout süresini 1 dakika yaptığımız için 1 dakika sonra Session nesnesinin değerinin null olması gerekmektedir. Bu süreyi bekledikten sonra yeni bir tarayıcı penceresi açıp sayfamıza girmek istersek aşağıdaki ekran görüntüsü ile karşılaşırız. Görüldüğü gibi ToplamZiyaretci değeri başlangıç değerine atanmıştır.



Ziyaretçi Sayısı 1

ASP.NET_SessionId **xn3edv4553zs4l450bqpdk45**

Ad Soyad

Session'a Ekle

[default](#)

[Sayfa2](#)

[Sayfa3](#)

Burada diğer bir durum daha vardır. Diyelim ki Timeout süresini 10 dakika yaptık. Uygulama çalıştırıldı ve bir kaç oturum açıldı. Sonra ise, 10 dakika dolmadan önce Timeout süresini 1 dakika yaptık. Ardından yeni oturumlar açtık. Son oturum açışından 1 dakika geçtikten sonra, önceden yaşam süreleri 10 dakika iken çalışan oturumlar durumlarını korumaya devam ederler.

Bazen istemci tarayıcılar Cookie desteğine sahip değildir veya bu özellikleri bilerek kapatılmıştır. Peki böyle bir durumda istemci bilgisayar, sunucuda kendisi için açılan oturum ile nasıl eşleştirilebilir? Bu sorunun çözümü ASP.NET_SessionId değerinin site içinde hareket edilen sayfaların linklerine eklenmesidir. Asp.Net bunu bizim için otomatik olarak yapmaktadır. Tek yapmamız gereken web.config dosyasında sessionState elemanında, Cookieless özelliğine true değerini atamaktır.

```
<sessionState
  mode="InProc" stateConnectionString="tcpip=127.0.0.1:42424"
  sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes"
  cookieless="true"
  timeout="20"
/>
```

Eğer uygulamamızı bu haliyle denersek sayfalar arasında gezinirken url bilgilerinin aşağıdaki gibi değiştiğini görürüz. Görüldüğü gibi ASP.NET_SessionId değeri otomatik olarak relative url bilgisinin önüne eklenmiştir.

default.aspx için;



Ziyaretçi Sayısı 1

ASP.NET_SessionId **44ha22u54xmmokrzhewcnx55**

Ad Soyad

Session'a Ekle

[default](#)

[Sayfa2](#)

[Sayfa3](#)

Eğer bu sayfadan diğer sayfalara geçiş yapar isek, oluşturulan ASP.NET_SessionId değerinin bu sayfaların url bilgisine de eklendiğini dolayısıyla Session'ın korunduğunu görürüz.



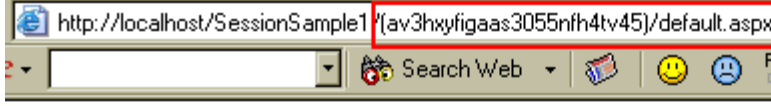
Burak

[default](#)

[Sayfa2](#)

[Sayfa3](#)

Ancak burada dikkat etmemiz gereken önemli bir nokta vardır. Linkler arasında ASP.NET_SessionId değerinin url bilgisi ile aktarılabilmesini istiyorsak relative linkleri kullanmak zorundayız. Yani Sayfa2 başlıklı LinkLabel nesnesinin NavigateUrl özelliğinin değerine, sayfa2.aspx (default.aspx ile aynı lokasyonda olduğu için) değerini vermemiz gerekir. Eğer buraya <http://localhost/SessionSample1/Sayfa2.aspx> gibi bir tam yol linki verirsek; bu linke tıkladığımızda default.aspx' teki oturum yok sayılacak ve yeni bir oturum açılmaya çalışılacaktır.



Ziyaretçi Sayısı 1

ASP.NET_SessionId **av3hxyfigaas3055nfh4tv45**

Ad Soyad

[default](#)

[Sayfa2](#)

[Sayfa3](#)



Server Error in '/SessionSample1' Applicati

Object reference not set to an instance of an object.

Görüldüğü gibi default.aspx sayfasını açtığımızda oluşan ASP.NET_SessionId değeri ile, buradan Sayfa2.aspx' e gittiğimizde URL bilgisine eklenen ASP.NET_SessionId değerinden farklıdır. Dahası burada, Sayfa2.aspx için yeni bir oturum söz konusu olduğundan, default.aspx içinde oluşturulan Session["Kimlik"] bilgisine buradan erişilemeyecektir. Bunun sonucuda da var olmayan bir nesneye erişmek istediğimizden hata almaktayız.

Bu makalemizde Session nesnesine kısaca değinmeye çalıştık. İzleyen makalemizde, Session nesnesi ile ilgili daha farklı konulara göz atmaya çalışacağız. Bu konular arasında ASP.NET_SessionId değerlerinin bir sql sunucusunda veya StateServer servisinde tutulması da yer almaktadır. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Derinlemesine Session Kullanımı – 2

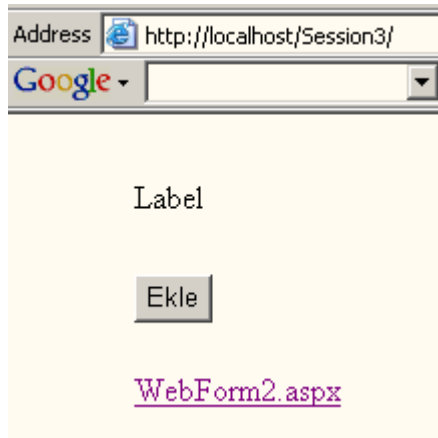
Değerli Okurlarım, Merhabalar.

Bir önceki makalemizde hatırlayacağınız gibi, Session nesnelerinin kullanımını incelemeye başlamıştık. Bu makalemizde ise, Session nesnelerinin nerelerde saklanabildiğine değinmeye çalışacağız. Varsayılan olarak Session nesneleri **In-Proc** (işlem içi) modunda saklanırlar. Yani web sayfasının çalıştığı asp.net work process' in içinde, dolayısıyla bu işlerin çalıştığı web sunucularındaki bellek alanlarında tutulurlar. Bu özellikle Session nesnelerine bilgi yazma ve okumada önemli bir avantajdır. Nitekim, erişim doğrudan ram üzerindeki bölgelere doğru olduğu için diğer bahsedeceğimiz modlara nazaran göreceli olarak oldukça hızlı bir erişim söz konusudur. Lakin, web sunucusunun başına bir şey gelmesi halinde (örneğin sunucunun bir anda restart olması gibi) bellekte tutulan tüm Session nesneleri bir anda kaybedilir. Bu da çok sayıda kullanıcının açtığı oturumlara ait bilgilerin tamamının

kaybolması anlamına gelmektedir. Bunu basit bir örnek ile gösterebiliriz. Aşağıdaki web uygulamasında, Session nesnesine bir değer atanmaktadır. Session'ın time-out süresi varsayılan halinde (Yani 20 dakika olarak) bırakılmıştır.

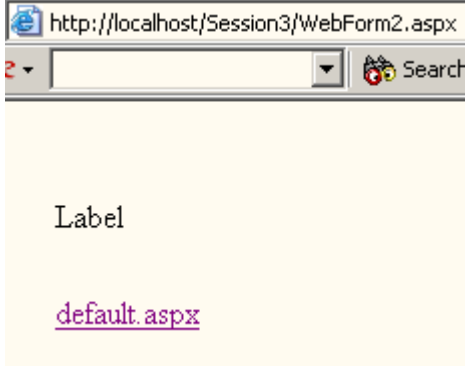
default.aspx kodları ve Form' un ekran görüntüsü;

```
private void Page_Load(object sender, System.EventArgs e)
{
    if(Session["Bilgi"]!=null)
    {
        Label1.Text=Session["Bilgi"].ToString();
    }
}
private void btnEkle_Click(object sender, System.EventArgs e)
{
    Session["Bilgi"]="Deneme";
    Label1.Text=Session["Bilgi"].ToString();
}
```

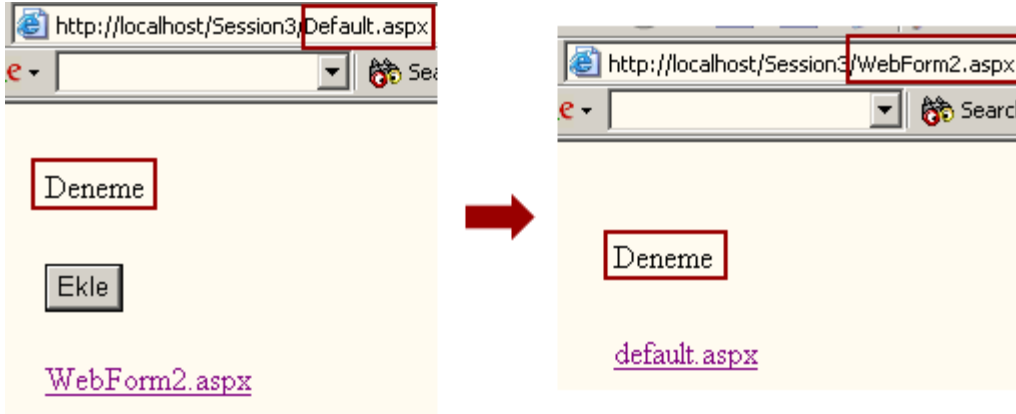


WebForm2.aspx kodları ve Form' un ekran görüntüsü;

```
private void Page_Load(object sender, System.EventArgs e)
{
    if(Session["Bilgi"]!=null)
        Label1.Text=Session["Bilgi"].ToString();
}
```



Bu örneği çalıştırdığımızda, default.aspx sayfasında Ekle butonuna basarsak Session nesnemize Deneme bilgisi eklenecektir. Eğer WebForm2.aspx sayfasına geçerseniz, Session bilgisinin Label kontrolüne yazıldığını görürüz.



Şimdi web uygulamamıza ait **Global.asax** dosyasında herhangi bir değişiklik yapıp uygulamamızı yeniden derleyelim. Ben örnek olarak pek bir anlam ifade etmeyen bir yorum satırı girdim ve uygulamayı yeniden derledim. Tabi bunu yaparken, web tarayıcımızda sayfalarımız açık halde bulunmalıdır.

```
protected void Application_Start(Object sender, EventArgs e)
{
    //YORUM SATIRIDIR.
}
```

Şimdi linklerimize tıklayarak sayfalar arasında tekrar gezindiğimizde, henüz time-out süresi dolmamış olan Session nesnelerinin kaybedildiğini ve Label kontrollerinde Session nesnesine ait içeriğin yazmadığını görürüz. Kısacası, In-Proc modunda olan (Yani işlem içi - In Process) Session nesneleri kaybedilmiştir. Elbetteki Session nesnelerinin bu şekilde kaybolması dışında da oluşabilecek istisnalar vardır. Örneğin sunucunun istem dışı bir şekilde kapanması gibi.

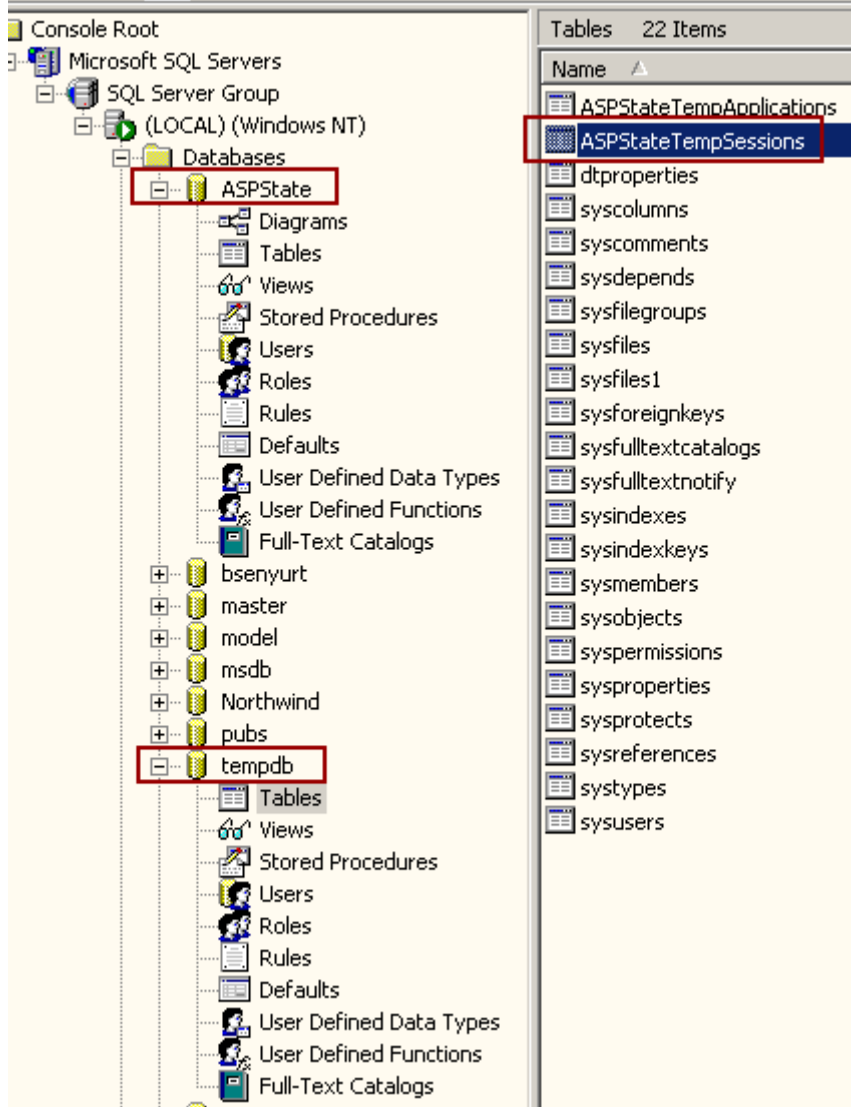
Asp.Net ile birlikte durum yönetiminde (state management), Session nesnelerinin saklanabilmesi için iki teknik daha geliştirilmiştir. Bu teknikler yardımıyla, durum nesnelerinin yukarıdaki gibi nedenlerden ötürü kaybolmalarının önüne geçilebilmektedir. Bu tekniklerden bir tanesi Session nesnelerinin bir **Sql Veritabanında** tutulduğu **SQLServer** modu, diğeri ise Session nesnelerinin **ASP.NET State Service Windows Servisinde** tutulduğu **StateServer** modudur. İlk olarak SQLServer modunu inceleyeceğiz.

SQLServer modunda, Session nesnesine ait tüm bilgiler bir Sql Sunucusunda bu iş için özel olarak hazırlanmış bir veritabanında tutulmaktadır. Böylece, Session nesnelerine ait içerik, istenen süre kadar (aylarca bile olabilir) fiziki bir disk bölgesinde saklanabilmektedir. Bu ayrıca, veritabanının başka bir sunucuda konuşturulmasıyla, Web Çiftliklerinin (**Web Farms**) yapısına uygun bir oluşumda imkan tanır. Böylece, Web Sunucusunda oluşabilecek aksaklıklardan doğacak sorunlar Sql Sunucusunu etkilemeyecek, dolayısıyla Session' lar korunmuş olacaktır. Elbette bu sistemin de dezavantajı vardır.



Session nesnelerini ayrı bir sunucudaki veri tabanında tutmak her ne kadar **güvenlik ve tutarlılık** açısından **yüksek performans** sağlasada, bilgilere erişimin In-Proc moda göre **daha yavaş** olmasında neden olur. Bu elbetteki verinin okunması veya yazılması için sürekli veritabanına doğru atılan turların bir sonucudur.

SQLServer modunda kullanılan ASPState isimli veritabanında Session nesnelerinin yazılma, silinme gibi işlemleri için kullanılan stored procedure' ler yer alır. Session nesnelerine ait asıl içerik ise tempdb isimli veritabanında yer alan tablolarda tutulmaktadır. Microsoft .NET Framework bu veritabanını ve içeriğini kurmak için gerekli Sql kodlarını içeren script dosyalarını içerir. Windows XP sistemlerinde bu dosyaya (**InstallSqlState.sql**) **D:\WINDOWS\Microsoft.NET\Framework\v1.1.4322** adresinden ulaşabilirsiniz. Bu sql script dosyasını **Sql Query Analyzer** ile çalıştırdığımızda, Sql Sunucusunda **ASPState** isimli bir veritabanı oluşturulduğunu görürüz. Ayrıca Session nesnelerini tutacak olan tablolarda **tempdb** veritabanı içerisine eklenirler.



Burada bizi asıl ilgilendiren kısım, **tempdb** veritabanında oluşturulan **ASPStateTempSessions** isimli tablodur. Nitekim bu tablo Session verilerini saklamak üzere kullanılmaktadır. Tablonun yapısı aşağıdaki şekilde görüldüğü gibidir.

	Column Name	Data Type	Length	Allow Nulls
▶	SessionId	char	32	
	Created	datetime	8	
	Expires	datetime	8	
	LockDate	datetime	8	
	LockDateLocal	datetime	8	
	LockCookie	int	4	
	Timeout	int	4	
	Locked	bit	1	
	SessionItemShort	varbinary	7000	✓
	SessionItemLong	image	16	✓

Dikkat edecek olursanız, **SessionId** isimli alan tablonun Primary Key alanıdır. Bu alanda, web sunucu tarafından otomatik olarak oluşturulan **ASP.NET_SessionId** değeri tutulmaktadır. Bunun dışında Session' ın yaratıldığı tarih, oturumun sona ereceği süre

bilgisi gibi veriler dışında Session nesnesinin içeriğinin saklanacağı iki önemli alan daha vardır. Bunlar **SessionItemShort** ve **SessionItemLong** alanlarıdır. Her iki alandan hangisinin kullanılacağı, Session nesnesine atanan verinin büyüklüğüne göre belirlenmektedir. İşte bu noktada karşımıza önemli bir sorun çıkar.



Bir DataSet içeriğini Session nesnesine aktardığımızda, SessionItemShort veya SessionItemLong alanlarından hangisi kullanılırsa kullanılsın, Session nesnesinin içerdiği veri nasıl olurda **tek bir alan içerisine** sığdırılabilir?

İşte burada bir önceki makalemizde bahsettiğimiz gibi Session nesnesinin taşıyacağı verinin serileştirilebilir olması gerekliliği ortaya çıkmaktadır. Böylece ister binary olarak ister XML olarak DataSet nesnesinin içeriği serileştirilebilir ve tek bir alan içerisine yazılıp okunabilir. Bu elbetteki Session ile taşımak istediğimiz her nesne örneği için geçerli bir durumdur. (Örneğin kendi yazdığımız bir sınıf için.)



Out-of-Proc (İşlem dışı) modlarda, Session nesnesine atanan nesnelerin **mutlaka serileştirilebilir (Serializable)** olmaları gerekmektedir.

Şimdi basit olarak yukarıda işlediğimiz örneğimizde kullandığımız Session nesnesini, **SQLServer** modunda saklayalım. Varsayılan olarak, Session nesneleri In-Proc modda tutulduklarından **web.config** dosyasında yer alan **sessionState** bölümünün standart içeriği aşağıdaki gibidir.

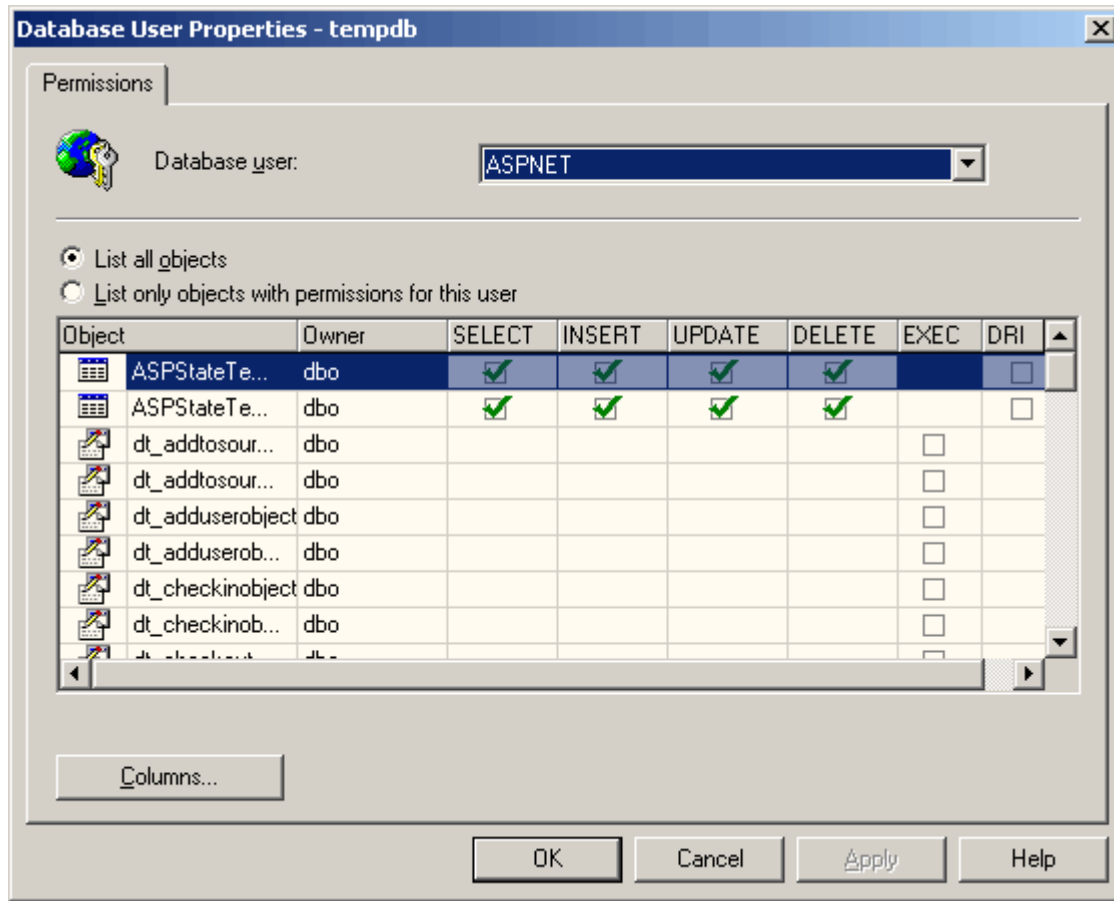
```
<sessionState
  mode="InProc"
  stateConnectionString="tcpip=127.0.0.1:42424"
  sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes"
  cookieless="false"
  timeout="20"
/>
```

Mode özelliğinin değeri varsayılan olarak InProc' tur. Yani, Session nesneleri işlem içinde tutulmaktadır. Session bilgilerini veritabanına yazabilmek için sessionState bölümünü aşağıdaki gibi düzenlememiz yeterli olacaktır.

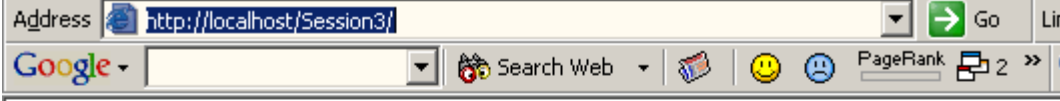
```
<sessionState
  mode="SQLServer"
  stateConnectionString="tcpip=127.0.0.1:42424"
  sqlConnectionString="data source=127.0.0.1;user id=sa;password=123456"
  cookieless="false"
  timeout="20"
/>
```

Burada dikkat etmemiz gereken en önemli nokta, **sqlConnectionString** özelliğinin aldığı bağlantı cümlecisidir. Bu özellikte, sql sunucusunun bulunduğu lokasyon **data source** ile belirtilmektedir. Varsayılan olarak Sql sunucusunun localhost üzerinde bulunduğu düşünülüğünden bu değer 127.0.0.1 ip değerini alır. Ancak **Web Çiftliği (Web-Farm)** gibi sistemlerde eğer Sql Sunucusunun bulunduğu adres farklı ise data source değerini bu adrese göre ayarlamamız gerekecektir. Diğer taraftan, ADO.NET'te olduğu gibi bağlantının yapılacağı veritabanı adının burada belirtilmesine gerek yoktur. Bunun sebebi SQLServer modunda Session nesnelerinin nereye yazılacaklarının zaten belli olmasıdır. Bir diğer husus ise, mutlaka ve mutlaka bağlantıyı belli bir kullanıcı adı ve şifresi üzerinden yapmamızın güvenlik açısından daha sağlıklı olacaktır.

Şunu da hatırlatmakta fayda var. InstallSqlState script' i her ne kadar session yönetimi için gerekli veritabanı düzenlemelerini yapsada, ASPNET kullanıcısına ASPState veritabanındaki stored procedure' leri çalıştırma ve tempdb içindeki ASPStateTempSessions ile ASPStateTempApplications tabloları için gerekli **Select, Insert, Update, Delete** komutlarını yürütebilme izinlerini vermemiz gerekiyor.



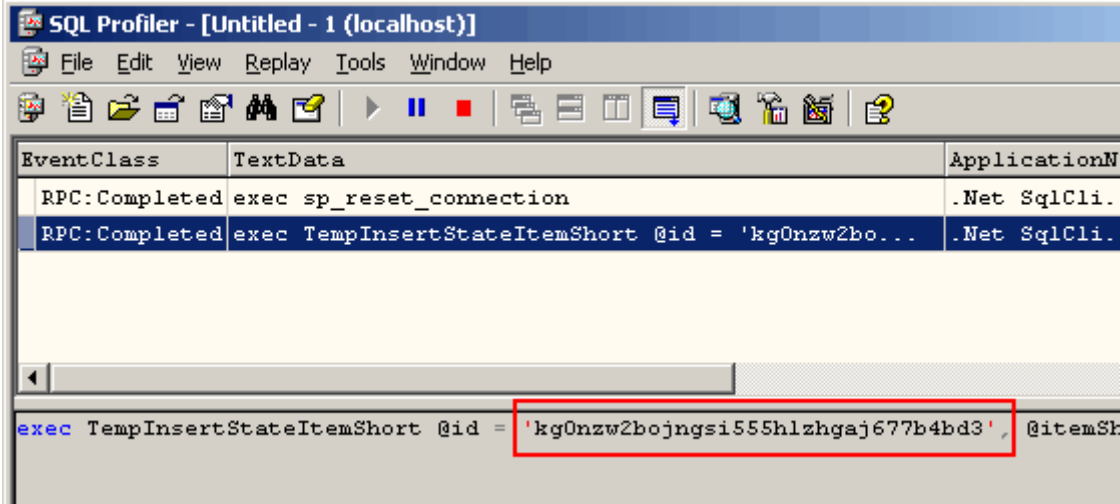
Aksi takdirde ASPNET kullanıcısının bu sp ve sql komutlarını çalıştırma yetkisi olmayacağından web uygulamamızda aşağıdaki benzer türden hata sayfaları ile karşılaşabiliriz.



Server Error in '/Session3' Application.

INSERT permission denied on object 'ASPStateTempSessions', database 'tempdb', owner 'dbo'.

Gerekli izinleride verdikten sonra artık uygulamamızı çalıştırabiliriz. Uygulamamızı çalıştırdığımız sırada eğer Sql Profiler ile arka planda olanları izlersek hemen bir sp' nin çalıştırıldığını ve sp' ye parametre olarak bir GUID' in aktarıldığını görürüz. Buradaki GUID, web sunucusu tarafından üretilen **ASP.NET_SessionId**' den başka bir şey değildir. Session nesnesi ilk kez yaratıldığından ilgili tabloya **INSERT** işlemini uygulayan bir sp çalışmaktadır.



Tam bu noktada ASPStateTempSessions tablomuza bakacak olursak, yukarıdaki sp'ye aktarılan Id' değerini SessionId alanına almış yeni bir satır oluşturulduğunu görürüz.

SessionId	Created	Expires	LockDate	LockDateLoc
kg0nzw2bojngsi555hlzhgaj677b4bd3	06.01.2005 20:30:14	06.01.2005 20:50:14	06.01.2005 20:30:14	06.01.2005 20:30:14

Elbetteki bu noktada, Session' ımıza henüz bir bilgi aktarmadığımız için tabloda yer alan ilgili alanlara herhangi bir bilgi yazılmamıştır. default.aspx sayfasında ekle butonuna basarsak, başka bir sp' nin bu kez var olan SessionId' li satırı güncellemek üzere çalıştırıldığını ve parametre olarakta encrypt edilmiş Session içeriğinin gönderildiğini görürüz.

```

TextData
exec sp_reset_connection

declare @P1 varbinary(7000) set @P1=0x1400000000000000FF declare @P2 bit set @P2=0 declare @P3 int set @P3=0
exec TempUpdateStateItemShort @id = 'kg0nzw2bojngsi555hlzhgaj677b4bd3', @itemShort = 0x1400000000000000FF, @P3 = 0

```

Dolayısıyla Session içeriği tabloda yer alan ilgili satıra (ASP.NET_SessionId değerine sahip olan satır) yazılmış olacaktır. Eğer makalemizin başındaki örneğimizde yaptığımız gibi, Global.asax dosyasında bir değişiklik yapıp uygulamayı yeniden derleyip time-out süresinden önce Session' ları okumak istersek, Session' lara ait değerlerin kaybolmadığını kolayca tespit edebiliriz. Session' ların yaşam süreleri dolduğunda otomatik olarak silindiklerini biliyoruz. InstallSqlState script' i ayrıca zaman aşımına uğramış Session' ların otomatik olarak kaldırılması için gerekli bir **job** nesnesinide sql sunucusuna yükler. (*Job nesnesinin çalışabilmesi için **Sql Server Agent** servisinin çalışıyor olması gerekmektedir.*)

Jobs	1 Item
Name	Ca
ASPState_Job_DeleteExpiredSessions	[U]

Şimdi aşağıdaki örneği inceleyelim. Bu örneğimizde, Session nesnesine bizim tanımladığımız bir nesne örneğini aktarıyoruz.

Personel isimli sınıfımız;

```
public class Personel
```

```
{
    private string mAd;
    private string mSoyad;

    public string Ad
    {
        get
        {
            return mAd;
        }
        set
        {
            mAd=value;
        }
    }

    public string Soyad
    {
        get
        {
            return mSoyad;
        }
        set
        {
            mSoyad=value;
        }
    }

    public Personel(string ad,string soyad)
    {
        mAd=ad;
        mSoyad=soyad;
    }
    public Personel()
    {
    }
}
```

default.aspx;

```
private Personel pOku;

private void Page_Load(object sender, System.EventArgs e)
{
    if(Session["Eleman"]!=null)
    {
        pOku=new Personel();
        pOku=(Personel)Session["Eleman"];
        Label1.Text=pOku.Ad+" "+pOku.Soyad;
    }
}

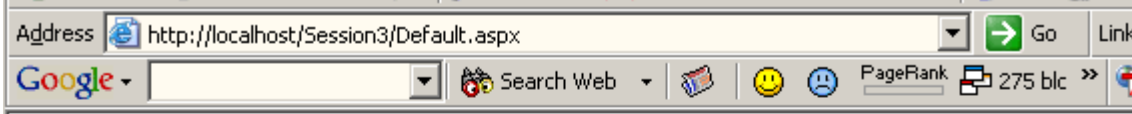
private void btnEkle_Click(object sender, System.EventArgs e)
{
    Personel p1=new Personel("Burak Selim","Şenyurt");
    Session["Eleman"]=p1;
}
```

WebForm2.aspx

```
private Personel pOku;

private void Page_Load(object sender, System.EventArgs e)
{
    if(Session["Eleman"]!=null)
    {
        pOku=new Personel();
        pOku=(Personel)Session["Eleman"];
        Label1.Text=pOku.Ad+" "+pOku.Soyad;
    }
}
```

default.aspx sayfasında Session nesnemize Personel sınıfından p1 isimli nesne örneğimizi aktarıyoruz. Her iki sayfada da Session nesnesinin içeriğini okurken, Personel sınıfından bir nesne örneğine açıkça bir dönüştürme işlemi yaptığımıza dikkat edelim. Çünkü Session nesnesi, kendisine atanan verileri object tipinde taşımaktadır. Şimdi default.aspx sayfamızı tarayıcı penceresinde açar ve Ekle butonuna basarsak aşağıdaki hata mesajını alırız.



Server Error in '/Session3' Application.

Unable to serialize the session state. Please note that non-serializable objects or MarshalByRef objects are not permitted when session state mode is 'StateServer' or 'SQLServer'.

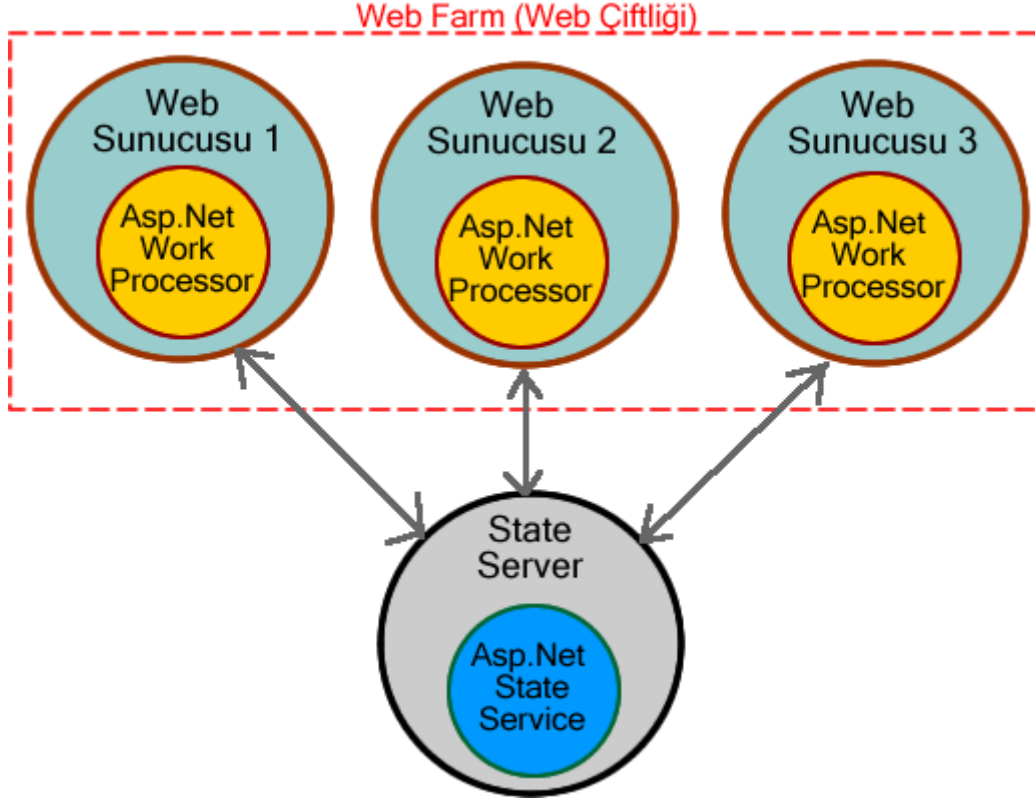
Sorun gayet açıktır. Personel sınıfımızın serileştirilebilir bir nesne olması gerekmektedir. Bu nedenle Personel sınıfımıza **Serializable niteliğini (Attribute)** eklememiz gerekiyor.

[Serializable]

```
public class Personel
{
    .
    .
    .
}
```

Şimdi Ekle butonuna tekrardan basarsak ve sayfalar arasında gezersek, Personel sınıfına ait nesne örneğinin başarılı bir şekilde taşındığını görürüz.

Session nesnelerini işlem dışından saklayabileceğimiz bir diğer seçenekte **ASP.NET State Service** isimli windows servisinin kullanılmasıdır. Bu kullanımda çoğunlukla, State Service başka bir sunucu üzerinde çalıştırılır ve diğer web sunucuları tarafından ortaklaşa kullanılır. Dolayısıyla, çalışan **Asp.Net work processor**' dan ayrı process' ler söz konusudur. Bu ayrı process' ler **State Server** üzerinde konuşlandırılır.



Session nesnelerini ASP.NET State Service ile kontrol edebilmek için öncelikle bu servisin çalıştırılması gerekir.

Application Layer Gateway Service	Started	Automatic	Local Service
Application Management		Manual	Local System
ASP.NET State Service	Started	Manual	Network S...
Automatic Updates	Started	Automatic	Local System
Background Intelligent Transfer Serv...	Started	Manual	Local System

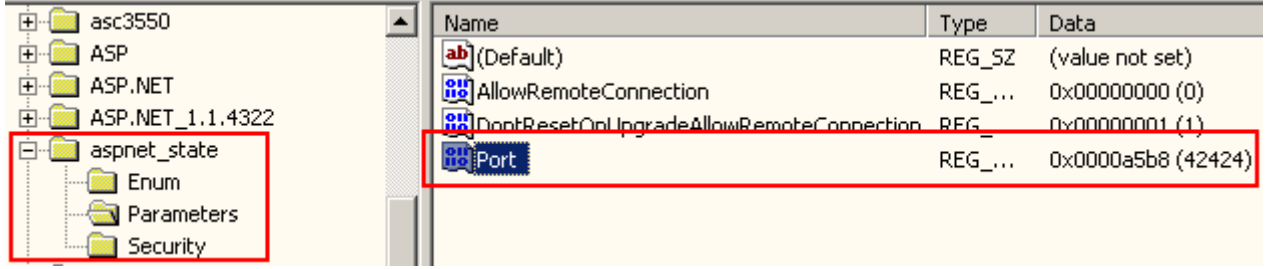
Servisin çalıştırılmasının ardından Web.config dosyasında da **sessionState** bölümünün özelliklerini aşağıdaki gibi değiştirmeliyiz.

```

<sessionState
  mode="StateServer"
  stateConnectionString="tcpip=127.0.0.1:42424"
  sqlConnectionString="data source=127.0.0.1;integrated security=SSPI"
  cookieless="false"
  timeout="20"
/>
  
```

StateServer modunda, State Server olarak kullanılacak sunucunun **tcpip adresi** ve ilgili **port numarası stateConnectionString** özelliğinde belirlenir. Biz burada local makineyi kullanıyoruz. Buradaki **42424** port numarası, ASP.NET State Service servisinin kullandığı varsayılan port numarasıdır. Dilersek bu numarayı değiştirmemiz mümkün. Bunun için, sistemdeki registry ayarlarına inmemiz gerekiyor. **Hot Key Local Machine** sekmesinde

\System\CurrentControlSet\Services\aspnet_state\Parameters altındaki **Port** elemanının değerini değiştirmemiz yeterlidir.



Bu değişikliklerden sonra Session nesnelerini ASP.NET State Service' in kontrolü altında tutulmak üzere kullanabiliriz. Bu servis yardımıyla tuttuğumuz Session nesneleri için de **serileştirilebilme şartı** aranmaktadır, burada hatırlatalım.

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Caching Mekanizmasını Anlamak – 1

Değerli Okurlarım, Merhabalar.

Bu makalemiz ile birlikte, web sayfalarının istemcilere daha hızlı bir şekilde ulaştırılmasında kullanılan tekniklerden birisi olan Caching (Ara Belleğe Alma) mekanizmasını incelemeye başlayacağız. Akıllıca kullanıldığı takdirde web uygulamalarında istemcilere nazaran göreceli olarak performans artışına neden olan Caching (Ara Belleğe Alma) mekanizması, teorik olarak bir web sayfasının tamamının ya da bir parçasının ara belleğe alınarak belli bir süre boyunca burada tutulması prensibini temel alarak çalışır. Asp.Net uygulamaları söz konusu olduğunda bir sayfanın tamamını, belli bir veri kümesini veya sayfa üzerindeki herhangi bir kontrolü ara belleğe alabiliriz. Buna göre Asp.Net uygulamalarındaki Caching (Ara Belleğe Alma) mekanizması aşağıdaki üç farklı tekniği destekler.

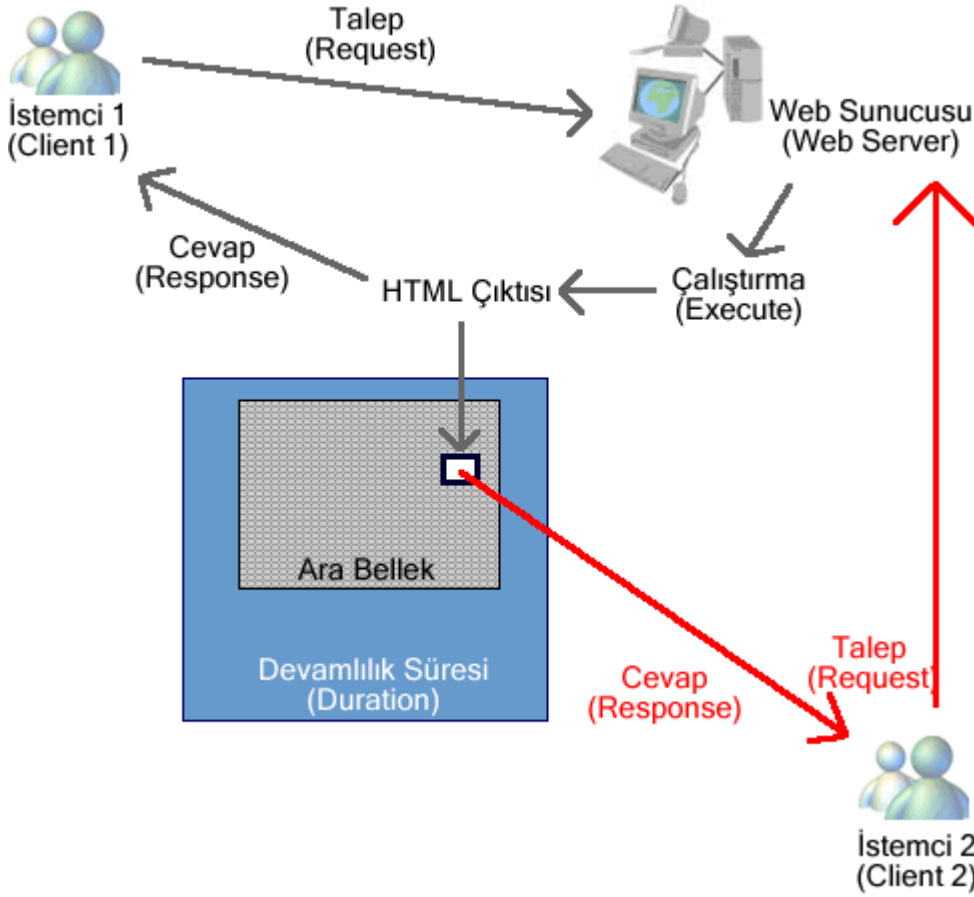
Asp.Net Uygulamaları İçin Caching (Ara Belleğe Alma) Teknikleri

1 - Output Caching (Çıktının Ara Belleğe Alınması)

2 - Data Caching (Verilerin Ara Belleğe Alınması)

3 - Fragment Caching (Parçaların Ara Belleğe Alınması)

Output Caching tekniğinde, bir aspx sayfasının tüm içeriği ara belleğe alınır ve belirli bir süre boyunca burada tutulur. Bu mekanizmanın çalışma biçimini aşağıdaki şekil ile daha kolay irdeleyebiliriz. Bu çalışma sistemi temel olarak diğer Caching mekanizmalarında da aynı şekilde işlemektedir. Değişen sadece ara belleğe alınan HTML görüntüsünün içeriği olacaktır.



İlk olarak, birinci istemci web sunucusundan bir aspx sayfasını talep eder. Web sunucusu sayfanın ilişkili kodlarını çalıştırarak bir çıktı üretir ve bu sayfa için Output Caching (Çıktının Ara Belleğe Alınması) aktif ise üretilen HTML sonuçlarını istemciye gönderir. Hemen ardından gönderilen HTML içeriğini belirtilen süre boyunca(duration) tutmak üzere sunucu üzerindeki ara belleğe alır.

Şimdi ikinci bir istemcinin devamlılık süresi (Duration) içerisindeyken aynı sayfayı sunucudan talep ettiğini düşünelim. Bu durumda web sunucusu, talep edilen sayfanın Cache (Ara Bellek) üzerinde olup olmadığına bakar. Eğer ikinci kullanıcı talebini, Devamlılık Süresi (Duration) içerisindeki bir zaman diliminde iletmış ise, web sunucusu istemciye sayfanın ara bellekteki hazır halini gönderir. Dolayısıyla sayfanın üretilmesinde çalıştırılan arka plan kodlarının hiç biri tekrardan yürütülmez. Bu elbetteki ikinci kullanıcının talep ettiği sayfa için daha kısa sürede cevap almasını sağlar.

Output Caching mekanizmasını bir aspx sayfasına uygulayabilmek için tek yapılması gereken OutputCache direktifinin sayfanın aspx kodlarının olduğu kısma eklemek yeterlidir.

```
<%@ OutputCache Duration="300" VaryByParam="None"%>
```

Burada, Duration özelliği ile sayfanın ilk talep edilışinden sonra 300 saniye (5 dakika) süre ile ara bellekte tutulacağını bildirmiş oluyoruz. VaryByName parametresini daha sonra inceleyeceğiz.

Şimdi dilerseniz, kısa bir örnek ile Output Caching mekanizmasının etkilerini incelemeye çalışalım. Basit bir web uygulaması geliştireceğiz. Bu uygulamada, Sql Sunucusunda yer alan Pubs isimli veri tabanındaki Authors isimli tabloya ait verileri bir DataGrid kontrolü içerisine aktaracağız.

default.aspx sayfamızın kodları;

```
private void Baglan()
{
    con=new SqlConnection("data source=localhost;initial catalog=pubs;integrated security=SSPI");
    cmd=new SqlCommand("SELECT * FROM authors",con);
    con.Open();
}
private void Doldur()
{
    dr=cmd.ExecuteReader(CommandBehavior.CloseConnection);
    dgVeriler.DataSource=dr;
    dgVeriler.DataBind();
    dr.Close();
}

private void Page_Load(object sender, System.EventArgs e)
{
    Baglan();
    Doldur();
}
```

Uygulama çalıştığında ve default.aspx sayfasının çıktısı üretildiğinde bu sayfanın son halinin bir kopyasında ara belleğe alınır. Yeni koypa ara bellekte 300 saniye (5 dakika) boyunca kalacaktır. Aynı sayfayı başka bir tarayıcı' da açtığımızda ya da aynı tarayıcı penceresinde iken ileri geri gittiğimizde sayfanın ara bellekte bulunan halini elde ederiz. Ancak bu örnekte dikkat edilmesi gereken bir nokta vardır. Bu durumu simule etmek için sayfayı refresh edelim. Bu durumda sayfanın içeriği aşağıdakine benzer olacaktır.



au_id	au_lname	au_fname	phone	address	city	state	zip	contract
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	True
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	True

Daha sonra Duration süresi dolmadan önce ilk satırın değerini Sql Sunucusu üzerinden değiştirelim. Bu durumda sayfada Output Caching uygulanmamış olsaydı, sayfayı bir sonraki talep ettiğimizde Load metodu çalışacak ve tablonun en güncel hali kullanıcıya sunulacaktı. Oysaki şimdi sayfayı talep ettiğimizde, yapılan değişikliğin görünmediğini farkederiz.

Değişiklik;

Object Browser Start Page default.aspx default.aspx.cs dbo.authors : T...le (BURKL.pubs)				
au_id	au_lname	au_fname	phone	address
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411
228-05-7766	Green	Cheryl	415 548-7723	590 Davis Ln

Yeni bir tarayıcı ile aynı sayfayı talep ettiğimizde;

http://localhost/Caching1/default.aspx								
au_id	au_lname	au_fname	phone	address	city	state	zip	contract
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	True
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	True

Bu işleyiş bazen büyük bir risk olabilir. Örneğin, sayfanın daha dinamik olduğu durumlarda ve hatta olay metodlarına cevap vermesi gerektiği durumlarda sayfanın tamamının ara belleğe alınması, aslında istemcilere sayfanın en güncel halinin gönderilmesi demek değildir. Bu gibi durumlarda çoğunlukla ya veri kümelerinin ya da sayfa üzerindeki belirli kısımların (kontrollerin) ara belleğe alınması tekniği tercih edilir. Bu teknikleri bir sonraki makalemizde inceleyeceğiz.

Bir sayfanın tamamını ara belleğe aldığımızda, sayfanın arabellekte yer alacak birden fazla kopyasına ihtiyaç duyduğumuz durumlar söz konusu olabilir. Çoğunlukla QueryString kullanımı sırasında başka sayfalara çeşitli parametreleri ve değerlerini göndeririz. İşte VaryByParam özelliği sayesinde sayfanın, gönderdiğimiz her parametre için ayrı ayrı veya sadece belirli parametreler için ayrı ayrı kopyalarını ara bellekte tutabiliriz.

Bu durumu daha yakından inceleyebilmek için pubs veritabanındaki titles tablosunu kullanacağımız bir örnek geliştirelim. Uygulamanın default.aspx sayfasında bu kez titles tablosunda yer alan her bir satır için title alanının değerlerini göstereceğiz. Kullanıcı her hangibir başlığa tıkladığında bu kitap ile ilgili detaylı bilgilerin olduğu başka bir sayfaya(detay.aspx) gidecek. Detay sayfası kitabın Primary Key değerini (title_id) QueryString parametresi olarak alacak. Bu durumda, detay sayfası için ara belleğe alma işlemini gerçekleştirebiliriz. Konuyu daha iyi anlayabilmek için örnek üzerinden adım adım gidelim. İlk olarak ana sayfamızın (default.aspx) aspx içeriğini aşağıdaki gibi değiştirmeliyiz.

```
<asp:DataGrid id="dgVeriler" style="Z-INDEX: 101; LEFT: 56px; POSITION: absolute; TOP: 88px" runat="server"
AutoGenerateColumns="False" Height="160px" Width="264px" BorderColor="#DEBA84" BorderStyle="None"
BorderWidth="1px" CellSpacing="2" BackColor="#DEBA84" CellPadding="3">
```

```

<FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5"></FooterStyle>
<SelectedItemStyle Font-Bold="True" ForeColor="White" BackColor="#738A9C"></SelectedItemStyle>
<ItemStyle ForeColor="#8C4510" BackColor="#FFF7E7"></ItemStyle>
<HeaderStyle Font-Bold="True" ForeColor="White" BackColor="#A55129"></HeaderStyle>
<PagerStyle HorizontalAlign="Center" ForeColor="#8C4510" Mode="NumericPages"></PagerStyle>
<Columns>
<asp:HyperLinkColumn DataNavigateUrlField="title_id" DataNavigateUrlFormatString="Detay.aspx?title_id={0}"
DataTextField="title" HeaderText="Başlık"></asp:HyperLinkColumn>
</Columns>
</asp:DataGrid>

```

Bu değişiklik sayesinde, dataGrid kontrolümüzden Detay.aspx sayfasına title_id alanının değerini parametre olarak gönderebileceğiz. Şimdi Detay.aspx sayfamının kodlarını aşağıdaki gibi geliştirelim.

```

private void Baglan(string Id)
{
    con=new SqlConnection("data source=localhost;initial catalog=pubs;integrated security=SSPI");
    cmd=new SqlCommand("SELECT title_id,title,price,pubdate FROM titles WHERE title_id='"+Id+"'",con);
    con.Open();
}

private void Doldur()
{
    dr=cmd.ExecuteReader(CommandBehavior.CloseConnection);
    dgDetaylar.DataSource=dr;
    dgDetaylar.DataBind();
    dr.Close();
}

private void Page_Load(object sender, System.EventArgs e)
{
    string id=Request.Params["title_id"].ToString();
    Baglan(id);
    Doldur();
}

```

Burada ara belleğe alma işlemini Detay.aspx sayfası üzerinde uyguluyoruz. Çünkü detay.aspx, dinamik olarak içeriği gelen parametre değerine göre değişen bir sayfadır. Biz gelen parametre değerine göre üretilen sayfaların html çıktılarını ara belleğe almak istiyoruz. İşte bunun için yine aspx kodlarının başına OutputCache direktifini aşağıdaki gibi eklememiz gerekiyor.

```
<%@ OutputCache Duration="300" VaryByParam="title_id"%>
```

Böylece detay.aspx sayfası her çalıştırıldığında gelen title_id değeri baz alınarak ara bellekte 300 saniye süreyle duracak olan html çıktıların oluşturulmasına imkan sağlıyoruz. Olayı şu şekilde irdelersek daha anlaşılır olacaktır;

Parametre tabanlı OutputCache işlemi	
İstemci title_id değeri BU1032 olan satırı detay.aspx sayfasından ister. Bu durumda detay.aspx sayfasının içeriği ara belleğe alınır. Bu sayfayı A olarak düşünelim.	detay.aspx in A koypası oluşturulur.
Başka bir İstemci title_id değeri MC2222 olan sayfayı talep eder. Bu durumda detay.aspx sayfasının yeni halinin kopyası ara belleğe alınır.	detay.aspx in B koypası oluşturulur.
Başka bir İstemci title_id değeri MC2222 olan sayfayı talep eder. Eğer duration süresi dolmadıysa talep edilen bu sayfanın çıktısı ara bellekte zaten olduğundan istemciye direkt olarak bu çıktı gönderilir.	detay.aspx in var olan B kopyası döner.
Başka bir İstemci title_id değeri BU1032 olan sayfayı talep eder. Eğer duration süresi dolmadıysa talep edilen bu sayfanın çıktısı ara bellekte zaten olduğundan istemciye direkt olarak bu çıktı gönderilir.	detay.aspx in var olan A kopyası döner.
<i>* Duration süreleri dolduğunda ise gelen talebe göre ara bellekteki sayfa çıktıları tekrardan oluşturulur.</i>	

Görüldüğü gibi VaryByParam özelliği ile, bir sayfanın kendisine QueryString ile gelen parametrelerinin değerine göre farklı ara bellek görüntülerini elde edebilmekteyiz. Bazı durumlarda talep edilen sayfaya birden fazla parametre gönderildiği veya hiç parametre gönderilmeden çağrıldığında olur. Bu durumu karşılamak için VaryByParam özelliğine * değerini atayabiliriz.

```
<%@ OutputCache Duration="300" VaryByParam="*" %>
```

Her ne kadar bir sayfanın Output Cache tekniği ile ara belleğe alınması avantajlı görünsede, özellikle olay kodlamalı sayfaların işleyişinde bu kullanım sorunlara yol açabilir. Her şeyden önce sayfa içinde postback' e neden olan kodlamalar var ise, sayfa ilk çağrıldıktan sonra ara belleğe alınacağından bu kod satırları duration süresi sonlanana kadar yürütülmeyecektir. Diğer yandan zaman zaman, sayfalarımızın içeriği dinamik olarak değişmek zorunda kalabilir. Bu gibi durumlarda da sayfanın tamamının ara belleğe alınması iyi bir yöntem değildir. Çözüm, sayfanın belirli parçalarının veya sayfadaki herhangi bir veri kümesinin(kümelerinin) ara belleğe alınmasıdır. Data Caching ve Fragment Caching tekniklerini bir sonraki makalemizde inceleyeceğiz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.