

2003 Yılı Yazıları

[0 - Stored Procedure Yardımıyla Yeni Bir Kayıt Ekleme - 08 Kasım 2003 Cumartesi](#)

[1 - Web Sayfalarında Stored Procedure Kullanımı - 12 Kasım 2003 Çarşamba](#)

[2 - Stored Procedure Yardımıyla Tablodan Kayıt Silme - 12 Kasım 2003 Çarşamba](#)

[3 - Overload Metodların Gücü - 13 Kasım 2003 Perşembe](#)

[4 - Transaction Kavramı - 17 Kasım 2003 Pazartesi](#)

[5 - Distributed Transactions - 19 Kasım 2003 Çarşamba](#)

[6 - Bir Form ve Kontrollerinin Elle Programlanması - 21 Kasım 2003 Cuma](#)

[7 - Params Anahtar Sözcüğünün Kullanımı - 30 Kasım 2003 Pazar](#)

[8 - DataSet ve WriteXml Metodunun Kullanımı - 30 Kasım 2003 Pazar](#)

[9 - Basit Bir Web Service Uygulaması - 30 Kasım 2003 Pazar](#)

[10 - Enumerators - 01 Aralık 2003 Pazartesi](#)

[11 - Struct Kavramı ve Class ile Struct Arasındaki Farklar - 04 Aralık 2003 Perşembe](#)

[12 - DataTable Sınıfını Kullanarak Programatik Olarak Tablolar Oluşturmak-1 - 04 Aralık 2003 Perşembe](#)

[13 - DataTable Sınıfını Kullanarak Programatik Olarak Tablolar Oluşturmak-2 - 05 Aralık 2003 Cuma](#)

[14 - DataColumn.Expression Özelliği ile Hesaplanmış Alanların Oluşturulması - 06 Aralık 2003 Cumartesi](#)

[15 - İlişkili Tabloları DataSet ile Kullanmak - 1 - 09 Aralık 2003 Salı](#)

[16 - İlişkili Tabloları DataSet ile Kullanmak - 2 - 10 Aralık 2003 Çarşamba](#)

[17 - SQL DMO İşlemleri - 12 Aralık 2003 Cuma](#)

[18 - DataView Sınıfı ve Faydaları - 15 Aralık 2003 Pazartesi](#)

[19 - DataGrid Denetimi Üzerinde Sayfalama\(Paging\) İşlemi - 16 Aralık 2003 Salı](#)

[20 - DataGrid Denetimi Üzerinde Sıralama\(Sorting\) İşlemi - 17 Aralık 2003 Çarşamba](#)

[21 - HashTable Koleksiyon Sınıfı - 18 Aralık 2003 Perşembe](#)

[22 - Stack ve Queue Koleksiyon Sınıfı - 19 Aralık 2003 Cuma](#)

[23 - Reflection Sınıfı İle Tiplerin Sırrı Ortaya Çıkıyor - 22 Aralık 2003 Pazartesi](#)

[24 - Bir Sınıf Yazalım - 23 Aralık 2003 Salı](#)

[25 - Virtual\(Sanal\) Metotlar - 25 Aralık 2003 Perşembe](#)

[26 - Kalıtım \(Inheritance\) Kavramına Kısa Bir Bakış - 25 Aralık 2003 Perşembe](#)

[27 - SqlDataReader Sınıfı 1 - 28 Aralık 2003 Pazar](#)

[28 - SqlDataReader Sınıfı 2 - 29 Aralık 2003 Pazartesi](#)

[29 - Boxing \(Kutulamak\) ve Unboxing \(Kutuyu Kaldırmak\) - 30 Aralık 2003 Salı](#)

Stored Procedure Yardımıyla Yeni Bir Kayıt Ekleme - 08 Kasım 2003

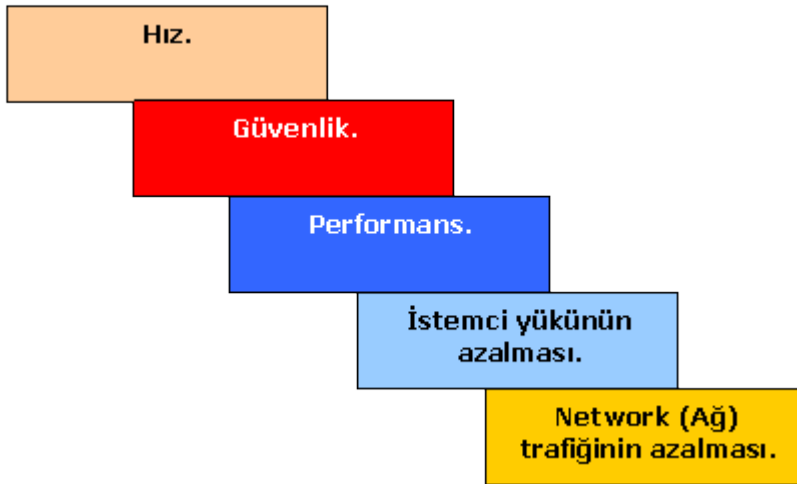
Cumartesi

ado.net, stored procedures,

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

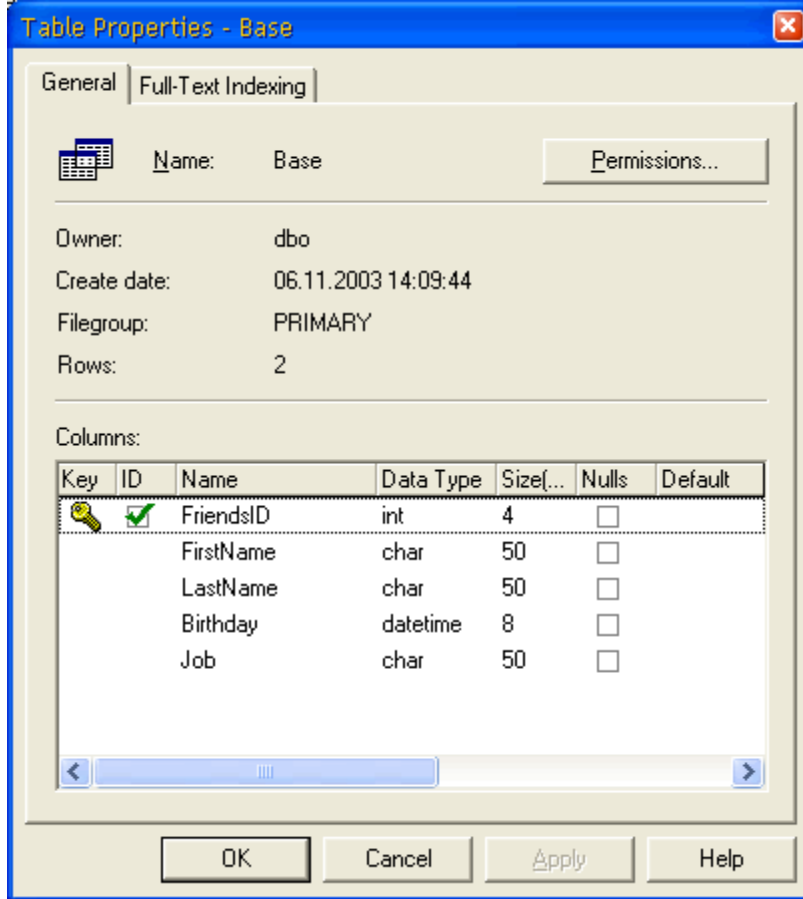
Öncelikle, Saklı Yordamlar hakkında kısa bir bilgi vererek hızlı bir giriş yapalım. Saklı yordamlar derlenmiş sql cümlecikleridir. Bunlar birer veritabanı nesnesi oldukları için, doğrudan veritabanı yöneticisi olan programda (örneğin Sql Server) yer alırlar. Bu nedenle veritabanınızı bir yere taşıdığınızda 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'lerden çok daha yavaş sonuç döndürür. Dolayısıyla Saklı Yordamlar özellikle çok katlı mimariyi uygulamak istediğimiz projelerde faydalıdır. Saklı Yordamların faydalarını genel hatları ile özetlemek gerekirse ;



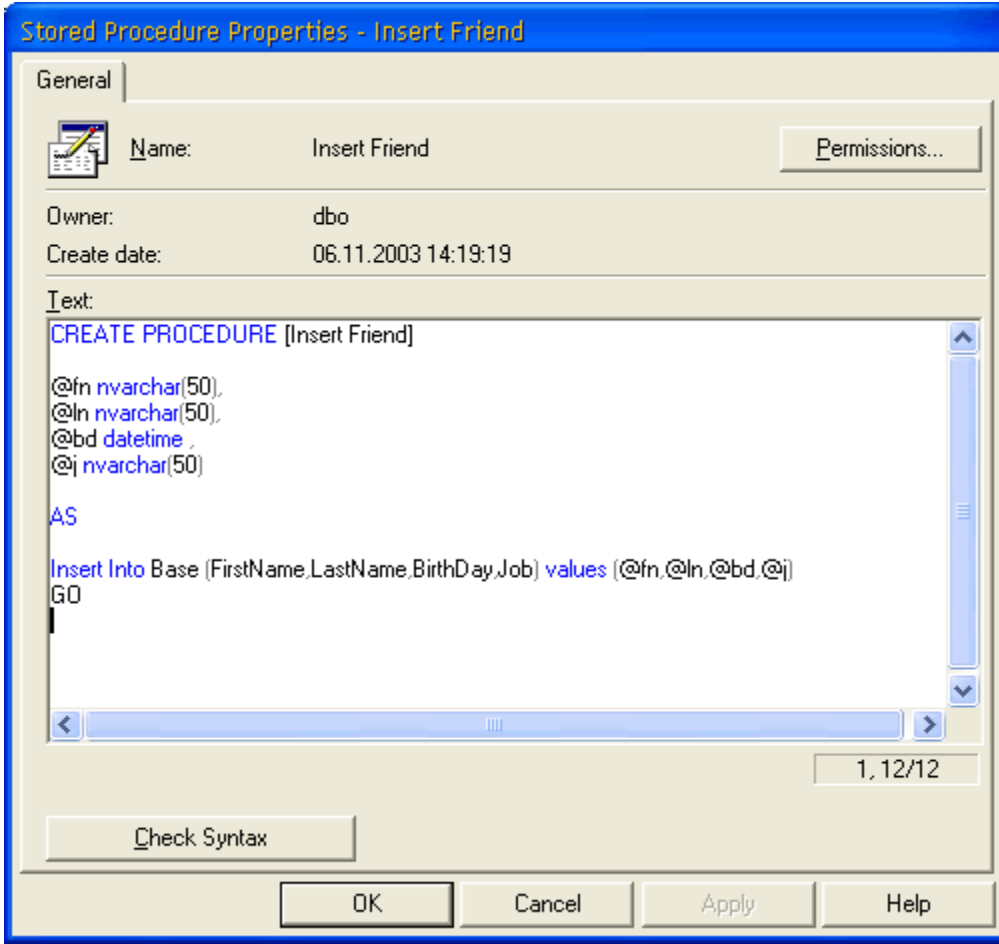
Şekil 1. Saklı Yordam Kullanmanın Avantajları.

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;

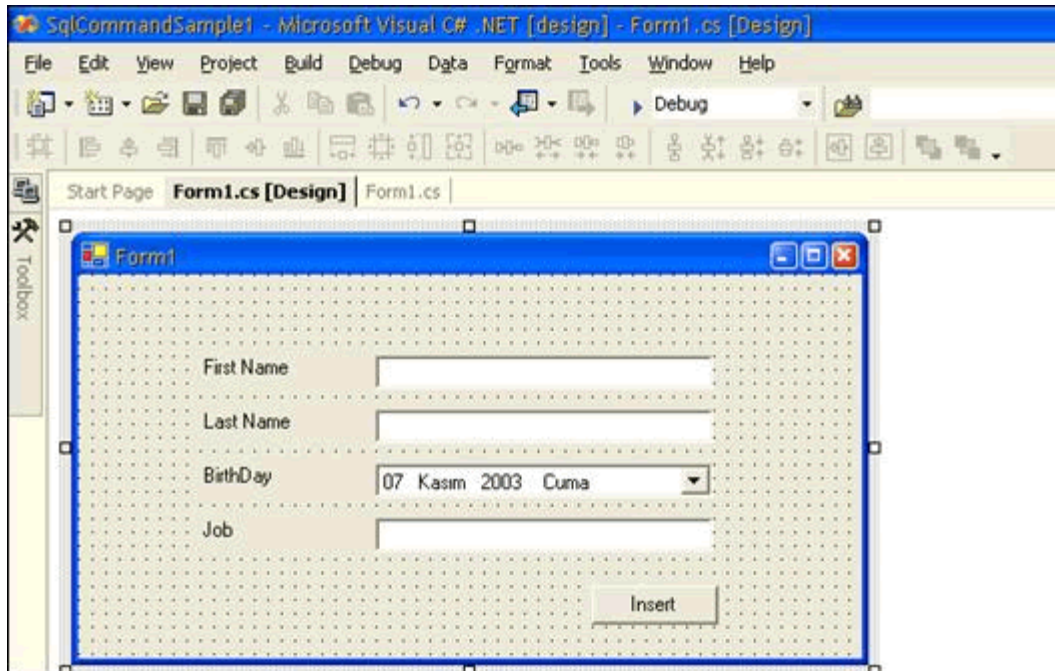
Parametre Adı	Veri Tipi	Veri Uzunluğu	Açıklama
@fn	Nvarchar	50	First Name alanı için kullanılacak.
@ln	Nvarchar	50	Last Name alanı için kullanılacak.
@bd	Datetime	-	BirthDay alanı için kullanılacak.
@j	Nvarchar	50	Job alanı için kullanılacak.

Tablo 1. Saklı Yordamımızda Kullanılan Giriş Parametreleri.

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

cümleciği 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 cümleciğine 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 nesnelerine 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 açiliyor. SqlCommand nesnesi ile ilgili ayarlamalara
geçiliyor. Komut SQL Server' da Friends database'inde yazili 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 olusturacagiz. Bunun için
SqlCommand nesnesininin parameters koleksiyonunun Add metodunu kullanıyoruz.
Parametreleri eklerken, parametre isimlerinin SQL Server'da yer alan Saklı Yordamlardaki
parametre isimleri ile ayni olmasına ve baslarına @ 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 olusan 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 paremetrelerimize degerlerini verelim.
paramFirstName.Value=txtFirstName.Text;
paramLastName.Value=txtLastName.Text;
paramBirthDay.Value=dtBirthDay.Text;
paramJob.Value=txtJob.Text;
// Böylece ilgili paremetrelere degerleri geçirilmis oldu. simdi komutu calistiralim.
cmdInsert.ExecuteNonQuery();
/* Böylece Saklı Yordamimiz, paremetrelerine 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 de tercih edilirler. Saklı Yordamları geliştirmek son derece kolaydır. İstedikini sql işlemi gerçekleştirilebilir. Satır silmek, satır aramak

gibi. Saklı Yordamlar ile ilgili bir sonraki makalemizde, tablolardan nasıl satır silebileceğimizi incelemeye çalışacağız. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Web Sayfalarında Stored Procedure Kullanımı - 12 Kasım 2003 Çarşamba

asp.net, stored procedures,

Bugünkü makalemden 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 Şekil2’ de görebilirsiniz.

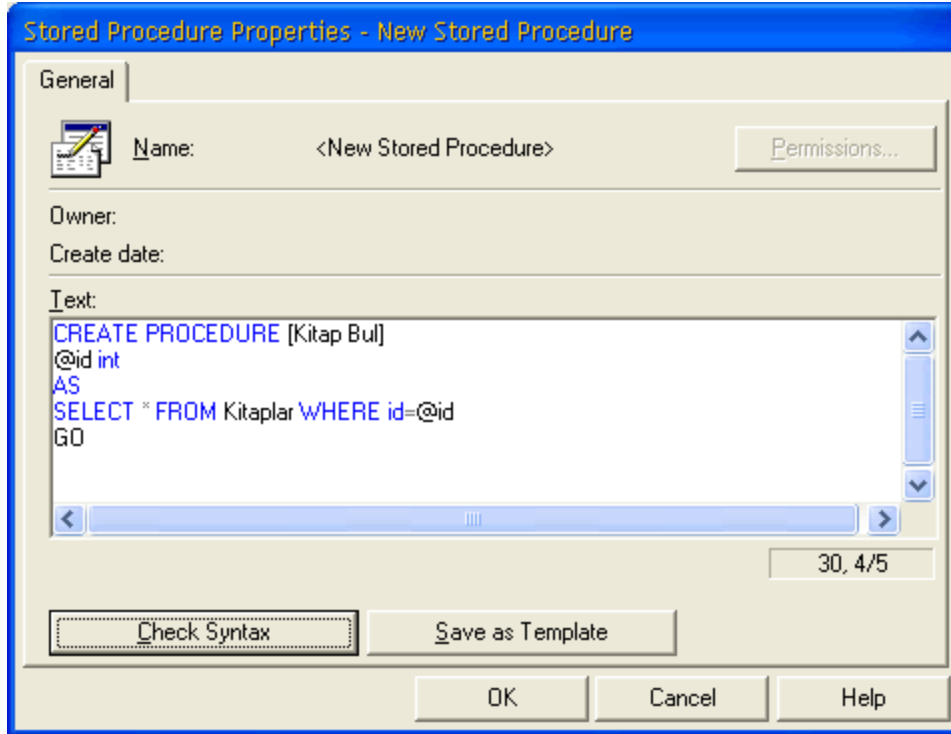
	Column Name	Data Type	Length	Allow Nulls
10	Kategori	int	4	
	Adi	nvarchar	50	
	Yazar	nvarchar	150	
	BasimEvi	nvarchar	100	
	BasimTarihi	nvarchar	50	
	Resim	nvarchar	4	✓
	Link	nvarchar	255	✓
	Fiyat	nvarchar	150	✓
		money	8	

Şekil 1. Kitaplar tablosunun alan yapısı.

ID	Kategori	Adı	Yazar	BasımEvi	BasımTarihi	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 Ust	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 K	Faruk Çubukçu	ALFA	2001	images/fk	<NULL>	<NULL>
8	WEB Programlama	ASP ile E-Ticaret Programcılığı	Stephen Walther, 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	Adım Adım Web Veritabanı Geliştirme	Jim Buyens	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 Sarac	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 Yıldırımöğlu	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 Çalkalp	BELLI DEĞİL	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 Balci	BALCI	2000	images/fk	<NULL>	<NULL>
27	Matematik Mühendisliği	Matematik Analiz Cilt 2	Mustafa Balci	BALCI	2000	images/fk	<NULL>	<NULL>

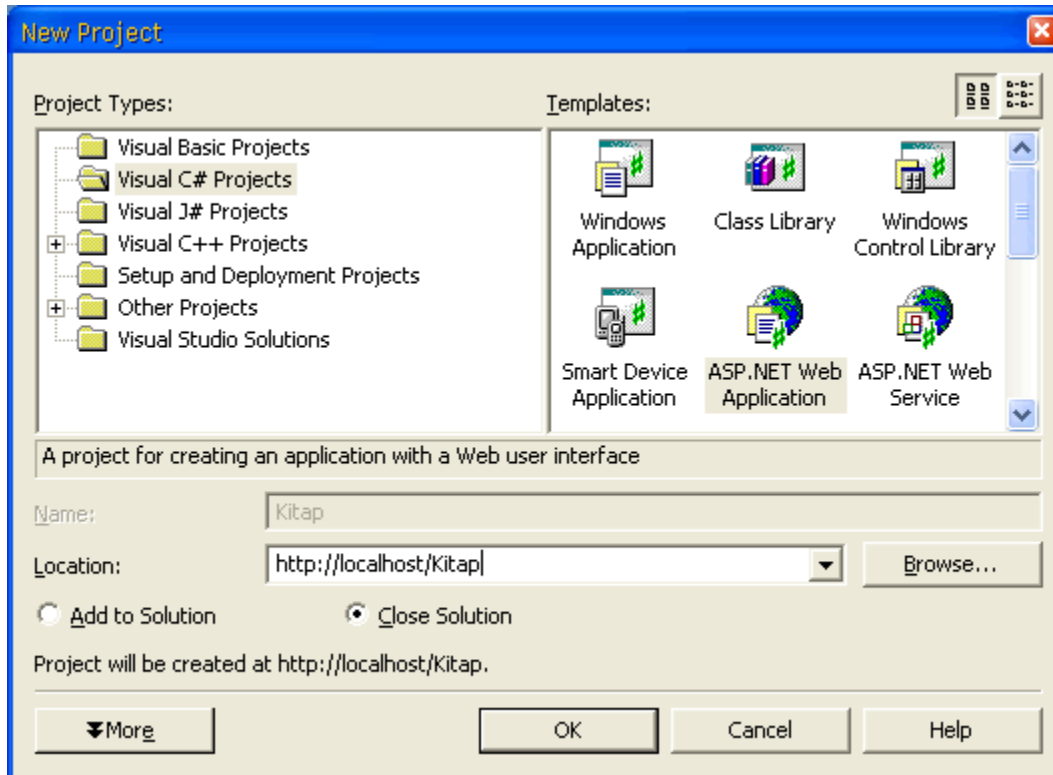
Şekil 2. Kitaplar tablosunun örnek verileri.

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 cümlecigi 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.



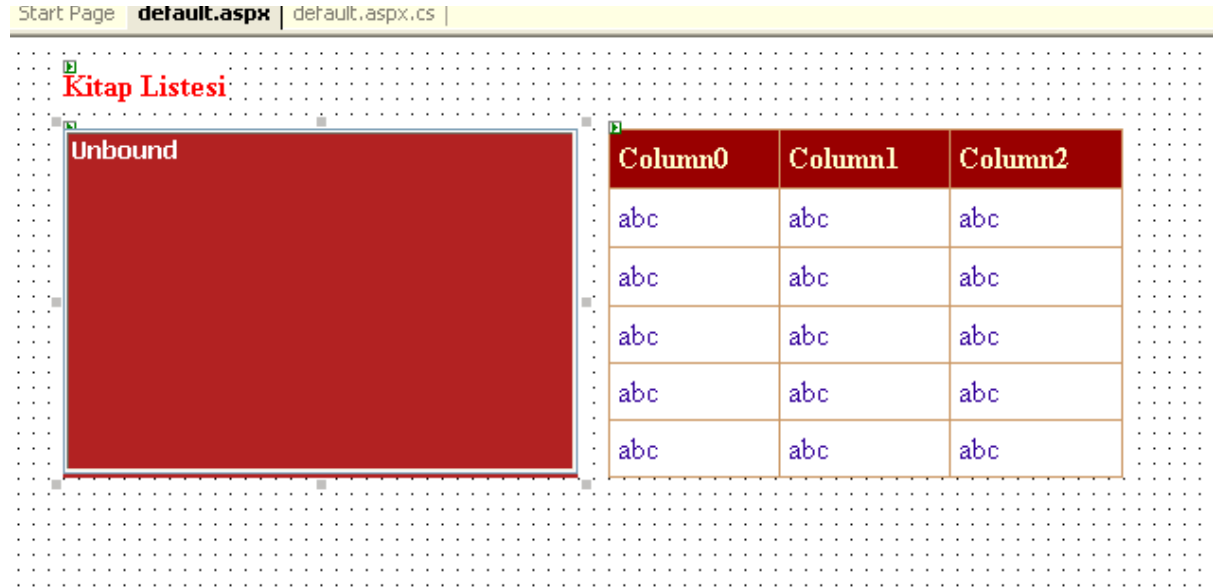
Şekil 3. Stored Procedure nesnemiz ve Sql ifadesi.

Sıra geldi uygulamamızda yer alan WebForm'umuzu oluşturmaya. Uygulamamızı C# dili ile oluşturmayı tercih ettiğimden New Project kısmında Visual C# Proejct bölümünü seçtim. Dikkat edicek olursanız, uygulamamız 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 alıcak. Alınan bu veri kümesi Adi alanına göre A'dan Z'ye sıralanmış olucak. 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 görünecek.

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.

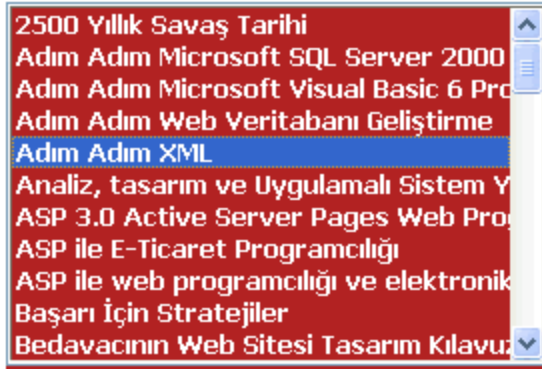
IstKitaplar.DataBind(); // Web Sayfalarında , verileri nesnelere bağlarken DataBind metodu kullanılır.

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

}
}

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

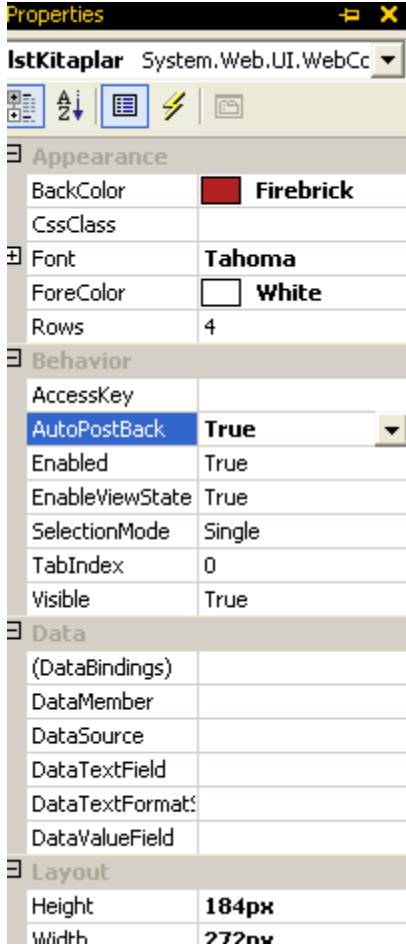
Kitap Listesi



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 IstKitaplar_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 IstKitaplar_SelectedIndexChanged olay procedure'ünün çalışmasını sağlamış oluyoruz. Nevarki 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	Adi	Yazar	Basın
59	Bilgisayar Programlama	Adım Adım Microsoft SQL Server 2000 Programlama	Rebecca M. Riordan	ARKA

Şekil 8. Sonuç.

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

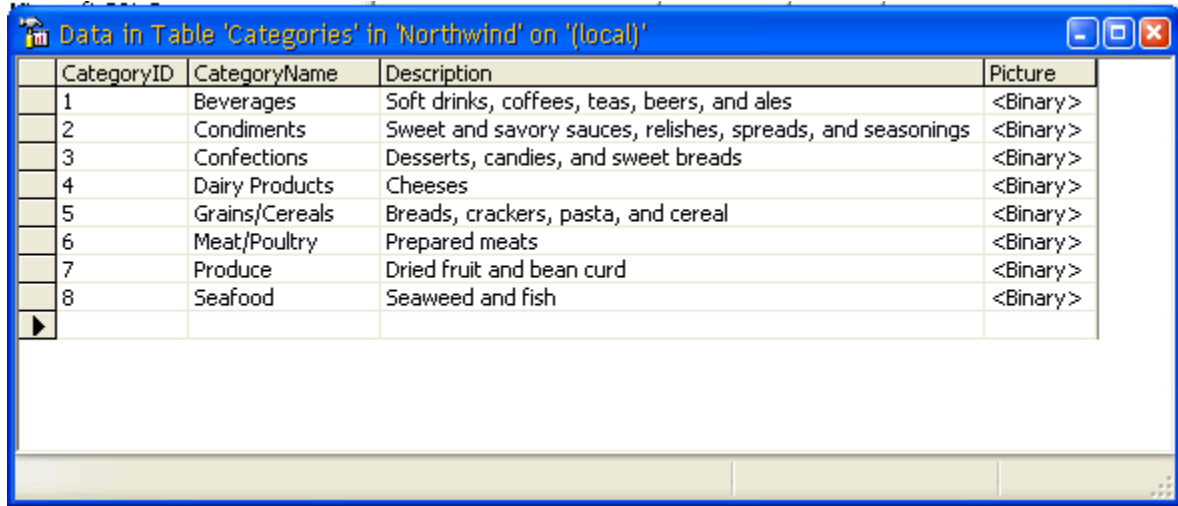
Stored Procedure Yardımıyla Tablodan Kayıt Silmek - 12 Kasım 2003 Çarşamba

ado.net, stored procedures,

Bugün ki makalemdede 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 gerçekçiliğidir. 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.

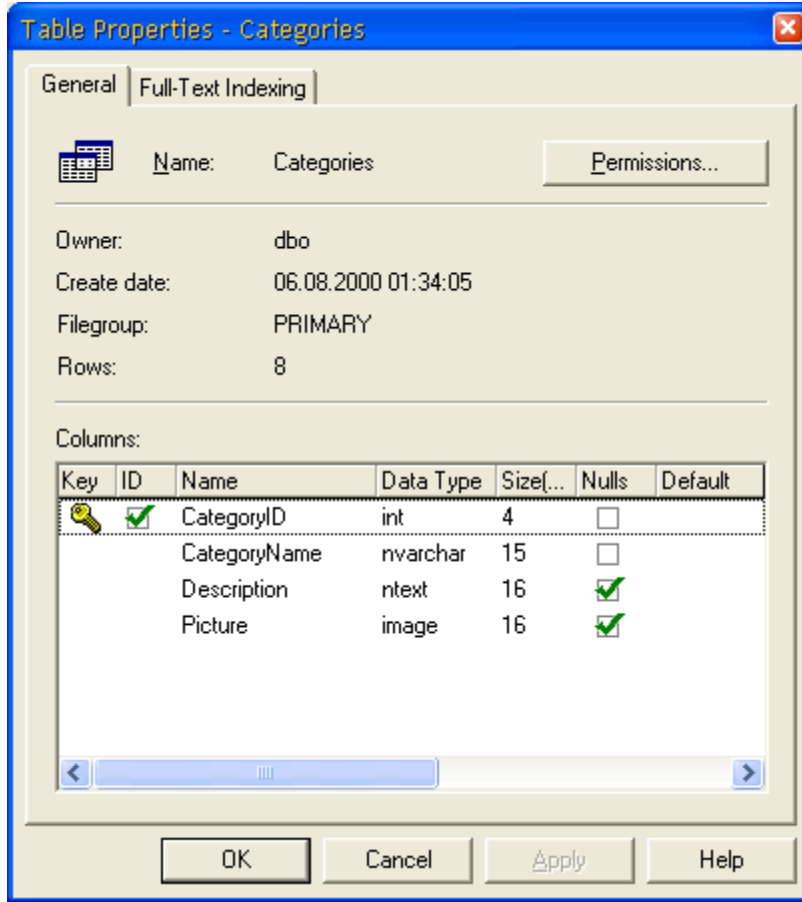
İşte bir önceki makalemde belirttiğim gibi Stored Procedure' leri kulanmamın en büyük amacı performans, hız ve güvenlik kriterlerinin önemidir. Dolayısıyla, örneklerimizi bu şekilde gerçek uygulamalara yakın tutarak, çalışırsak daha başarılı olacağımız inancındayım.Evet bu kadar laf kalabalığından sonra 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>

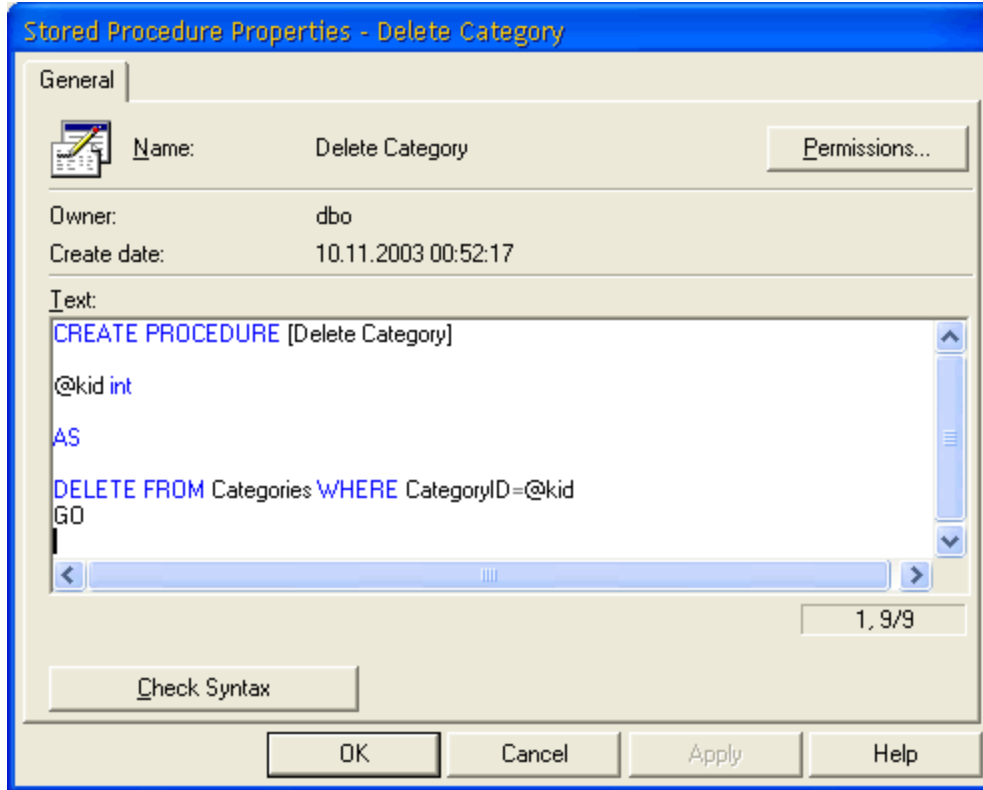
Şekil 1. Categories tablosunda yer alan veriler.

Tablomuzun yapısını da kısaca incelersek ;



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.

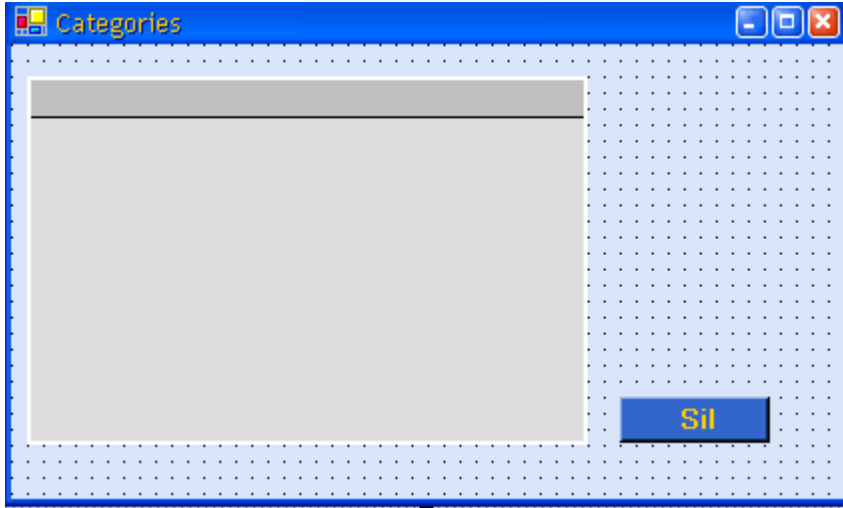


Şekil 3. Stored Procedure Kodları

```
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) cümlecisi 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 var ya) 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 engellemek 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
çalıştığı bilgisayar üzerindeki bellekte sakliyacaktır. 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 tanımladık.
```

/* Aşağıdaki satır ile yarattığımız SqlDataAdapter nesnesine bir Select sorgusu eklemiş oluyoruz. Bu sorgu sonucu dönen değer kümesi, SqlDataAdapter nesnesinin Fill metodunu kullandığımızda DataTable' in içeriğini hangi veriler ile dolduracağımızı belirtecek önemli bir özelliktir. Bir SqlDataAdapter nesnesi yaratıldığında, SelectCommand özelliğine SqlCommand türünden bir nesne atanarak bu işlem gerçekleştirilir. Burada aslında, SelectCommand özelliğinin prototipinden dolayı new anahtar sözcüğü kullanılarak bir SqlCommand nesnesi parametre olarak verilen select cümlecigi ile oluşturulmuş ve SelectCommand özelliğine atanmıştır.

*

```
* public new SqlCommand SelectCommand
* {
* get;
* set;
* }
```

* Prototipten de görüldüğü gibi SelectCommand özelliğinin tipi SqlCommand nesnesi türündendir. Bu yüzden new SqlCommand("....") ifadesi kullanılmıştır.

* */

```
da.SelectCommand=new SqlCommand("SELECT * FROM Categories");
da.SelectCommand.Connection=conNorthwind; /* Select sorgusunun alistirilacagi
baglanti belirlenir.*/
```

da.FillSchema(dtbCategories,SchemaType.Mapped);/* Burada dataTable nesnemize, veritabanında yer alan Categories isimli tablonun Schema bilgilerinde yüklüyoruz. Yani primaryKey bilgileri, alanların bilgileri yükleniyor. Bunu yapmamızın sebebi, Stored Procedure ile veritabanındaki Categories tablosundan silme işlemini yapmadan önce , bellekeli tablodan da aynı satırı silip dataGridView içindeki görüntünün ve DataTable nesnesinin güncel olarak kalmasını sağlamak. Nitekim silme işleminde DataTable nesnesinden seçili satırı silmek için kullanacağımız Remove metodu PrimaryKey alanının değeri istemektedir. Bunu verebilmek için tablonun PrimaryKey bilgisinde belleğe yani bellekteki DataTable nesnesine yüklenmiş olması gerekir. İşte bu amaçla Schema bilgilerinde alıyoruz*/

da.Fill(dtbCategories);/* Burada SqlDataAdapter nesnesinin Fill metodu çağırılır. Fill metodu öncelikle SelectCommand.CommandText in değeri olan Select sorgusunu alıştırır ve dönen veri kümesini dtbCategories isimli DataTable nesnesinin bellekte referans ettiği alana yükler. Artık bağlantıyıda kapatabiliriz.*/

```
conNorthwind.Close();
/* Simdi dataGridView nesnemize veri kaynagi olarak DataTable nesnemizi gösterecegiz.
Böylece DataGridView, Categories tablosundaki veriler ile dolucak. */
dgCategories.DataSource=dtbCategories;
}
```

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

/* Şimdi silme işlemini gerçekleştireceğimiz Stored Procedure'e DataGridView 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 dataGridView'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 alıştıracak 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 uyarmalıyız. 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.*/
```

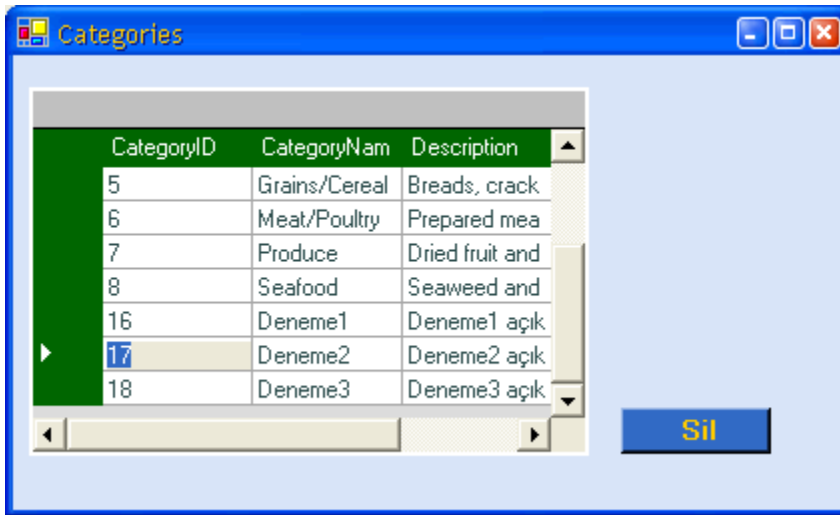
```
DialogResult result;
result=MessageBox.Show("CategoryID : "+selectedCategoryID.ToString()+" Bu satırı
silme istediğinizden emin misiniz?", "Sil", MessageBoxButtons.YesNo,
MessageBoxIcon.Question, MessageBoxDefaultButton.Button1);
if (result==DialogResult.Yes ) /*Eğer kullanıcının cevabı evet ise 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 satır bilgisi 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 silme
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.

Data in Table 'Categories' in 'Northwind' on '(local)'				
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>	
*				

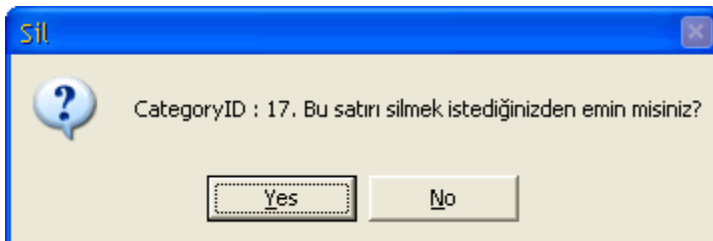
Şekil 5. Categories tablosuna 3 yeni kayıt ekledik.

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.



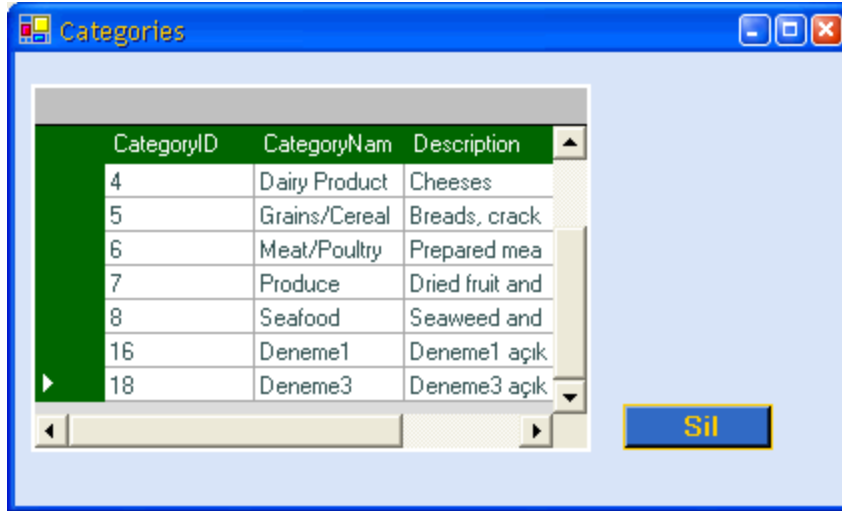
Şekil 6. Load Procedure'ünün çalıştırılmasından sonraki görünüm.

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, p	<Binary>
6	Meat/Poultry	Prepared meats	<Binary>
7	Produce	Dried fruit and bea	<Binary>
8	Seafood	Seaweed and fish	<Binary>
16	Deneme1	Deneme1 açıklamas	<Binary>
18	Deneme3	Deneme3 açıklamas	<Binary>
*			

Şekil 9. Sonuç.

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

Overload Metodların Gücü - 13 Kasım 2003 Perşembe

c#, overloading, method overloading, oop,

Değerli Okurlarım, Merhabalar.

Bu makalemdede 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 tamamıyla 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	Metodun Geri Dönüş Tipi
Parametrenin Tipleri	

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)  
        {
```

```
        }
```

```
    }
```

```
}
```

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

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.

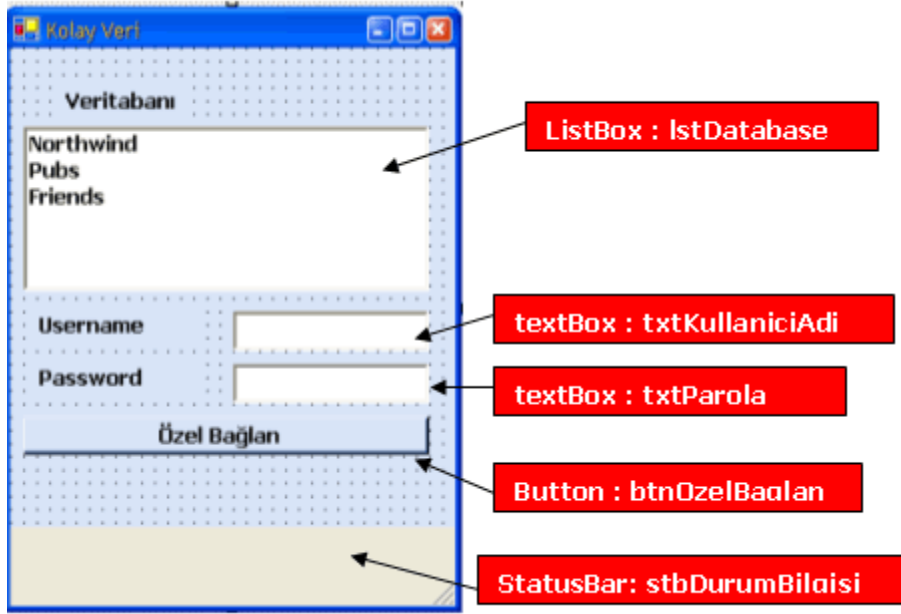
Yapıcı metodlarda overload edebiliriz. Bu da oldukça önemli bir noktadır. Bunu zaten .NET ile program geliştirirken sıkça kullanırız. Örneğin SqlConnection sınıfından bir nesne örneği yaratmak istediğimizde, bunu yapabileceğimiz 2 overload edilmiş yapıcı metod olduğunu görürüz. Bunlardan birisi aşağıda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



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

Dolayısıyla bizde yazdığımız sınıflara ait constructorları overload edebiliriz. Şimdi derseniz 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 olacak. Bu sınıfın üç adet yapıcısı olacak. 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 tasarım 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 nesnemizin 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'in durumunu tutacak ve sadece bu class içinde geçerli olan bir string değişken tanımladık. private anahtar kelimesi değişkenin sadece bu class içerisinde yaşayabilceğini belirtir. Yazmayabiliriz de, 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 yazdığımız aşırı yüklenmiş ilk yapıcı
metoda gelince. Burada yapıcıyı, 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
```



```
kullanici,sifre;  
    kullanici=txtKullaniciAdi.Text;
```

```
sifre=txtParola.Text;
```

```
veritabaniAdi=lstDatabase.SelectedItem.ToString();
```

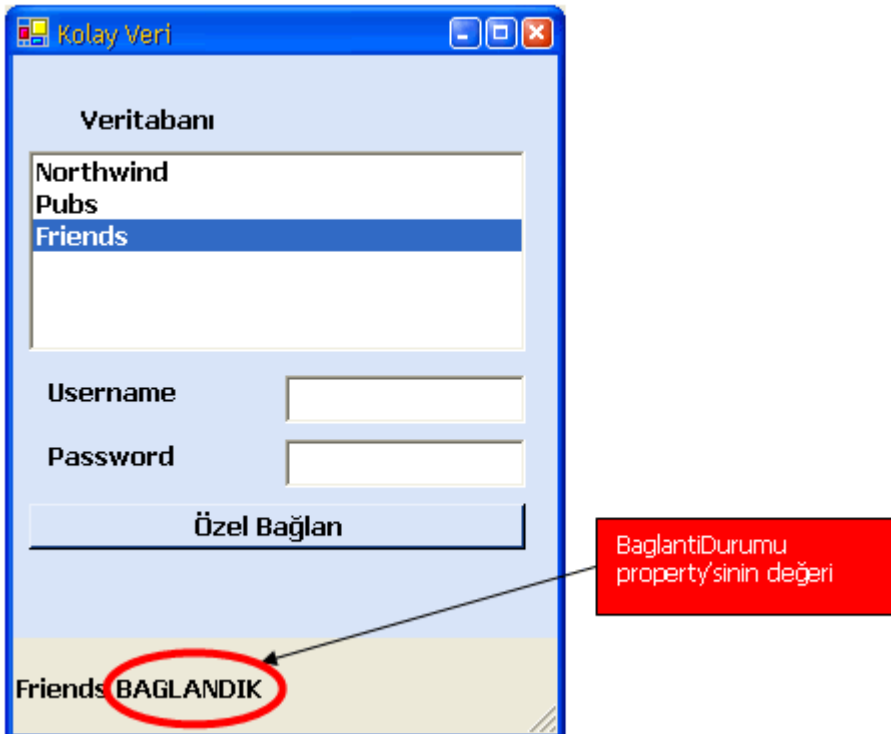
```
KolayVeri kvOzel=
```

new KolayVeri(veritabaniAdi,kullanici,sifre); /* Burada ise diğer aşırı yüklenmiş yapıcımızı 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 kullaniciAdi, string parola)
```

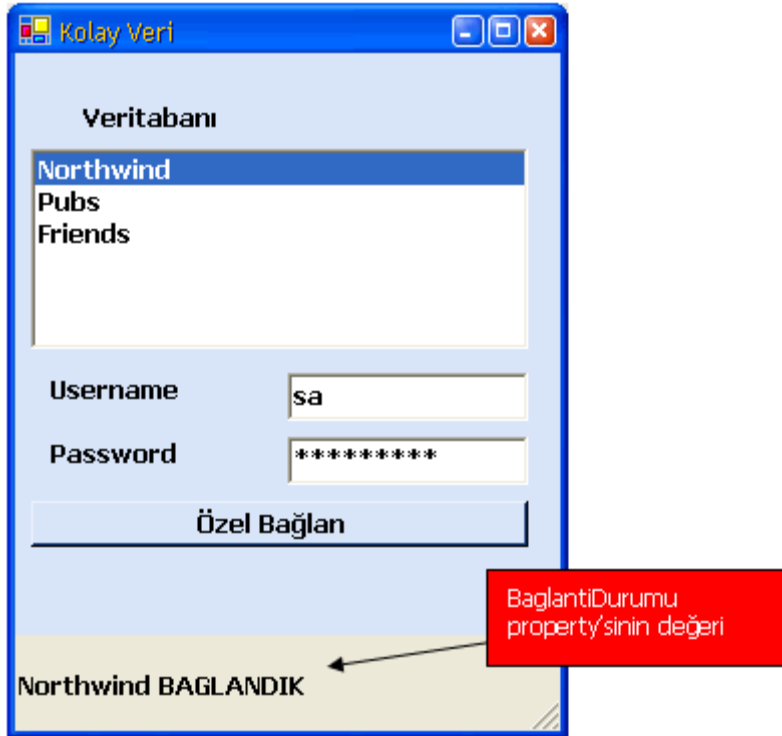
```
stbDurumBilgisi.Text=lstDatabase.SelectedItem.ToString()+" "+kvOzel.BaglantiDurumu+"  
User:"+kullanici;  
    }  
}
```

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



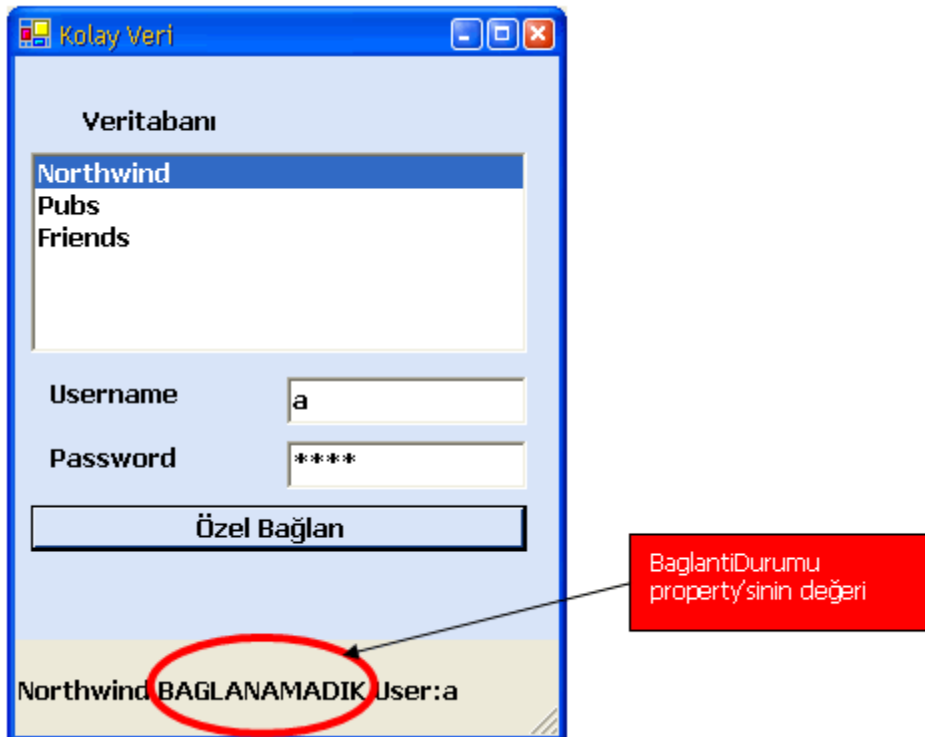
Şekil 6. Listboxta tıklanan veritabanına bağlandıktan sonra.

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. Yanlık 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 - 17 Kasım 2003 Pazartesi

ado.net, transactions,

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 tablolardan da aynı verileri sildiğimiz işlemleri başlatabiliriz. Dikkat edicek 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 diğeride 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 ihraç birimimize rapor ediyoruz. İhraç 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 parçacıdır 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 takib eden iş parçacıklarının doğru şekilde işlemesine 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 maleklenmediğini düşünecek olursak 1000 televizyon kalması gerekir. 1001 televizyon veya 999 televizyon değil. İşte bu verilerin tutarlılığını gerektirir.

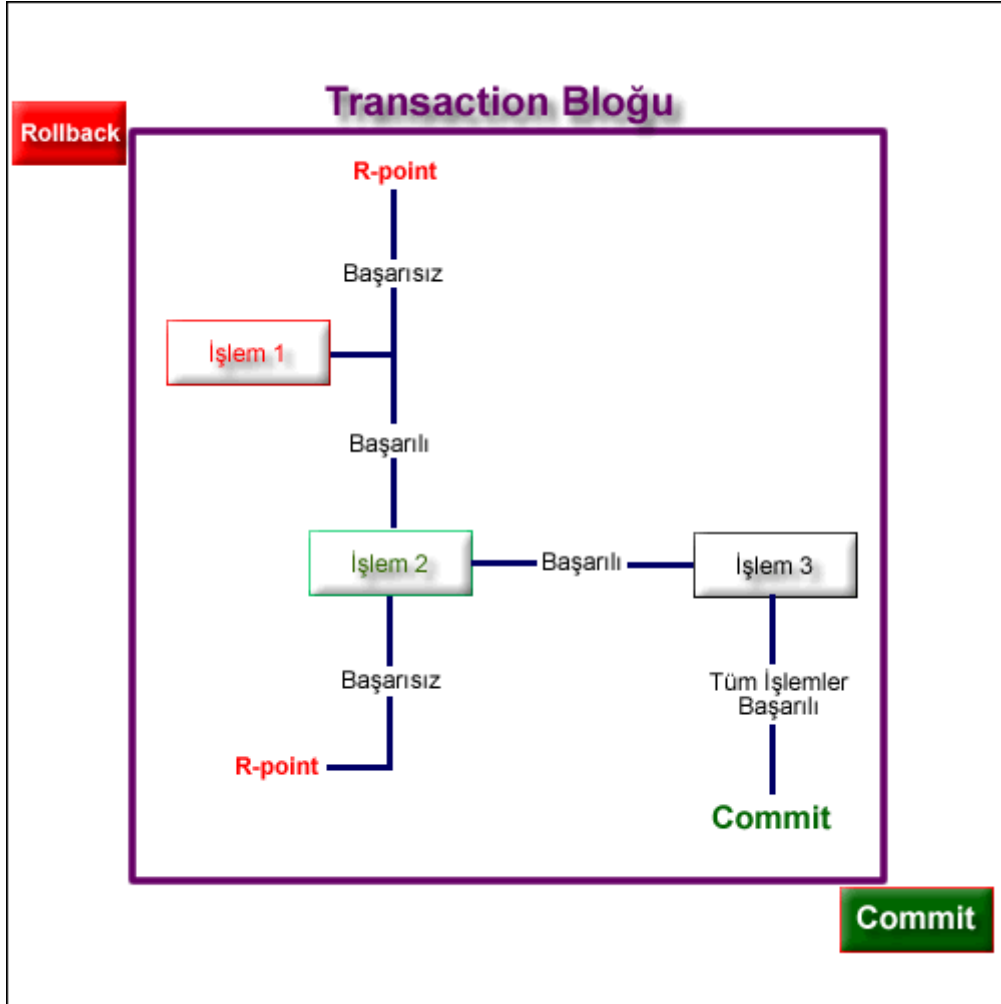
İşte bu nedenlerde ötürü Transaction kavramı ortaya çıkarmıştır. Bu kavrama göre aslında bahsedilen tüm iş parçakları kusursuz olarak başarılı olduklarında "işlem tamam" denebilir. İşte bizde veritabanı uygulamalarımızı geliştirirken, bu tip iş parçacıklarını bir Transaction bloğuna alırız. Şimdi ise 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 hangisinde 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 işlemişiine 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

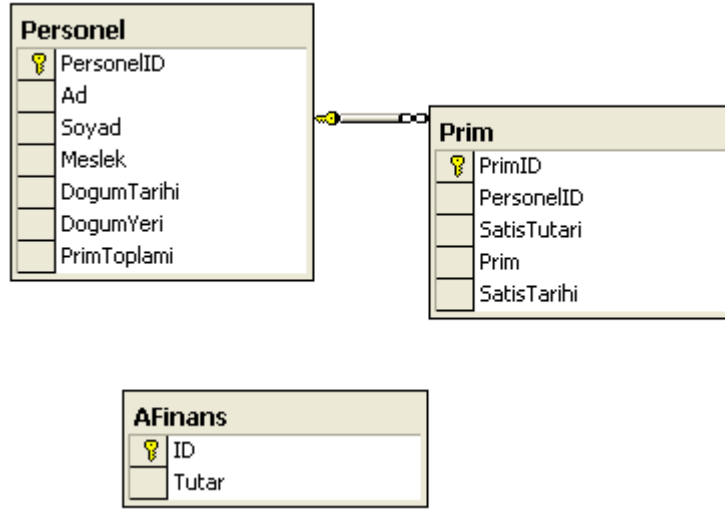
yazabileceğimiz Transaction' lardan bahsetmek istiyorum..NET içerisinde SqlClient sınıfında yer alan nesneleri Transaction nesneleri kullanılarak bu işlemi gerçekleştirebiliriz. Ben SqlTransaction nesnesini ele alacağım. Bu nesneyi oluşturmak için herhangi bir yapıcı metod yoktur. SqlDataReader sınıfında 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. İstanbulMerkez 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. Ayrıca AFinans tablosunda bir başlangıç tutarımızın olması gerekiyor.

PersonelID	Ad	Soyad	Meslek	DogumTarihi	DogumYeri	PrimToplami
1	Hasmet	Samet	Matematik Mühendisi	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 cümleciği 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'sını 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. */

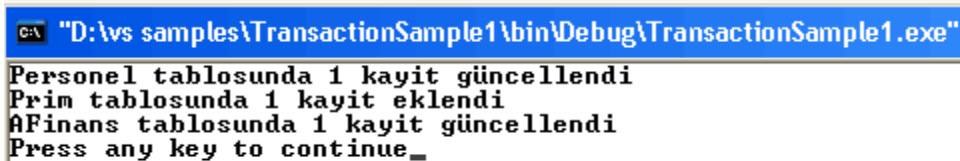
SqlTransaction tr;
conIstanbulMerkez.Open();
tr=conIstanbulMerkez.BeginTransaction();
double satisTutari=150000000000;
double primTutari=satisTutari*0.01;
/* Şimdi, Personel tablosundaki PrimToplami alanın değerini
primTutari değişkeninin değeri kadar arttıracak Stored Procedure'ü çalıştıracak
SqlCommand 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 parçasıdır." */
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 parametre ayarlamaları yapılıyor
ve yine Transaction nesnesi belirlenerek bu komut nesneside Transaction bloğu içerisine bir
iş parçası 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 olduğu ana kadar olan tüm işlemler
        iptal edilir.*/
        {
            Console.WriteLine(hata.Message+" Nedeni ile işlemler iptal
edildi");
            tr.Rollback();
        }
        finally /* hata oluşsada oluşmasada açık bir bağlantı var ise bunun
        kapatılmasını garanti altına almak için finally bloğunda bağlantı nesnemizi Close metodu ile
        kapatırız.*/
        {
            conIstanbulMerkez.Close();
        }
    }
}
}

```



```

D:\vs samples\TransactionSample1\bin\Debug\TransactionSample1.exe
Personel tablosunda 1 kayıt güncellendi
Prim tablosunda 1 kayıt eklendi
AFinans tablosunda 1 kayıt güncellendi
Press any key to continue_

```

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 Mühendi	04.10.1980	Istanbul	1500000000

Şekil 6. Personel tablosunda PrimToplami alanının değeri arttırıldı.

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 tablondaki 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'da olduğu gibi.

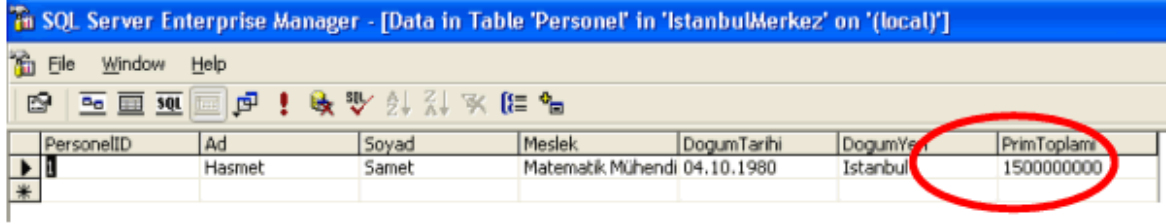
```
SqlCommand cmdPrimBilgisiGir=new SqlCommand("Prim Bisi Gir",conIstanbulMerkez);
```

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 alanının 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 - 19 Kasım 2003 Çarşamba

ado.net, transactions, 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

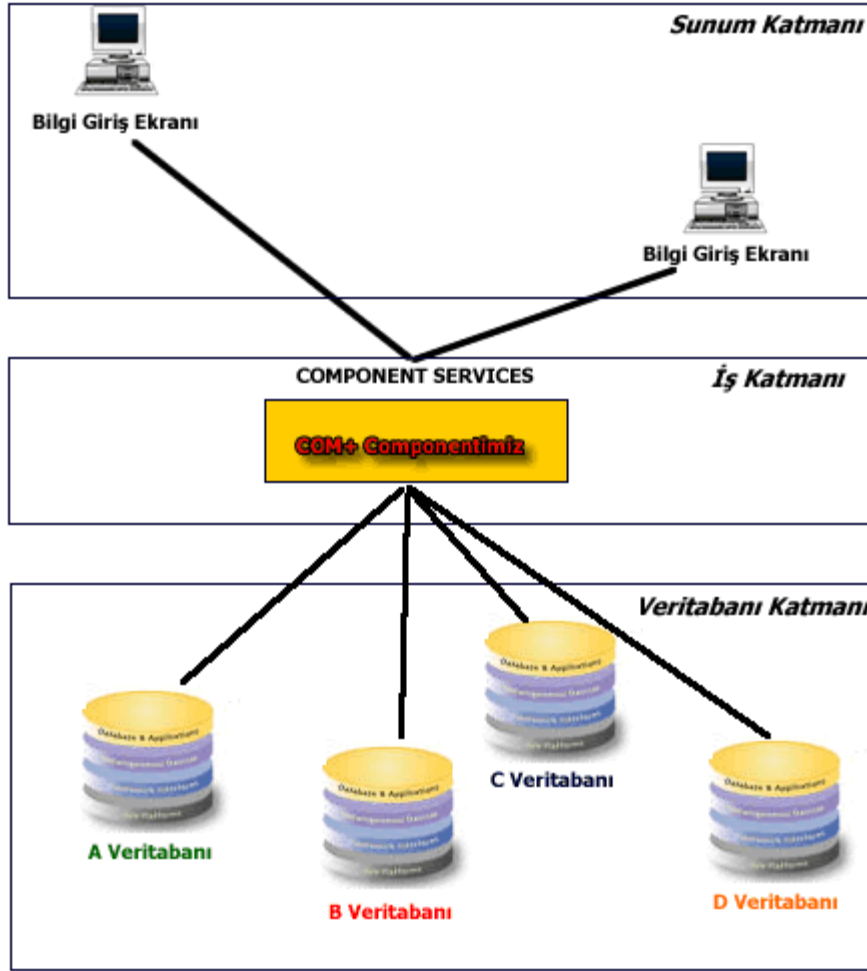
Component Services'i kullanmamız gerekmektedir. Özellikle, çok katlı mimari dediğimiz iş uygulamalarında Dağıtık Transaction'ları çok sık kullanırız. Burada Dağıtık Transaction'lar başka componentler tarafından üstlenilir ve Sunu Katmanı ile Veritabanları arasındaki işlemlerin gerçekleştirilmesinde rol 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 sonlandırırlar. 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 bağlanır , gerekli düzenlemeler bir Transaction içerisinde gerçekleştirilir ve sorunsuz bir transaction tüm iş parçacıkları ile teyid edilerek gerekli düzenlemeler veritabanlarına yansıtılır.

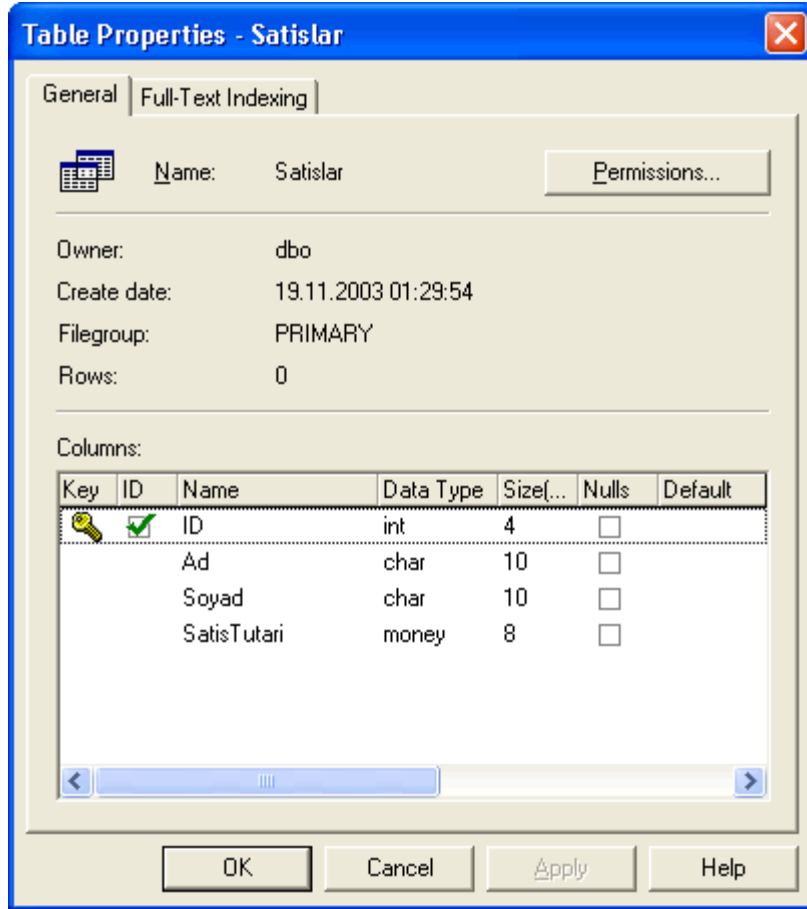
En basit haliyle 3 katlı mimaride, Sunum Katmanı ile Veritabanları arasındaki transaction işlemlerini COM+ Component'leri ile gerçekleştirebiliriz. Bu component'ler Windows 2000'den itibaren Component Service olarak adlandırılan bir servis altında yer almaktadırlar. Elbetteki bu component'i biz geliştireceğiz. 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 zihnimizde daha berrak canlanacağı kanısındayım.



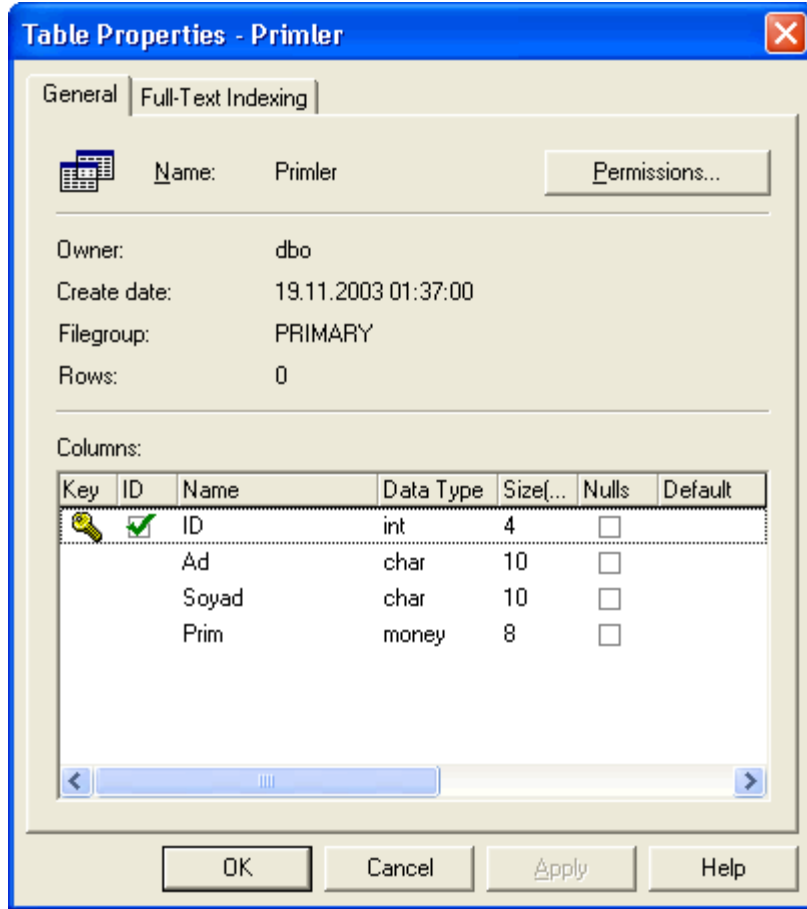
Şekil 1. 3 Katlı Mimari için COM+ Kullanımı

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 uygulamamızda 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. Satislar 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.

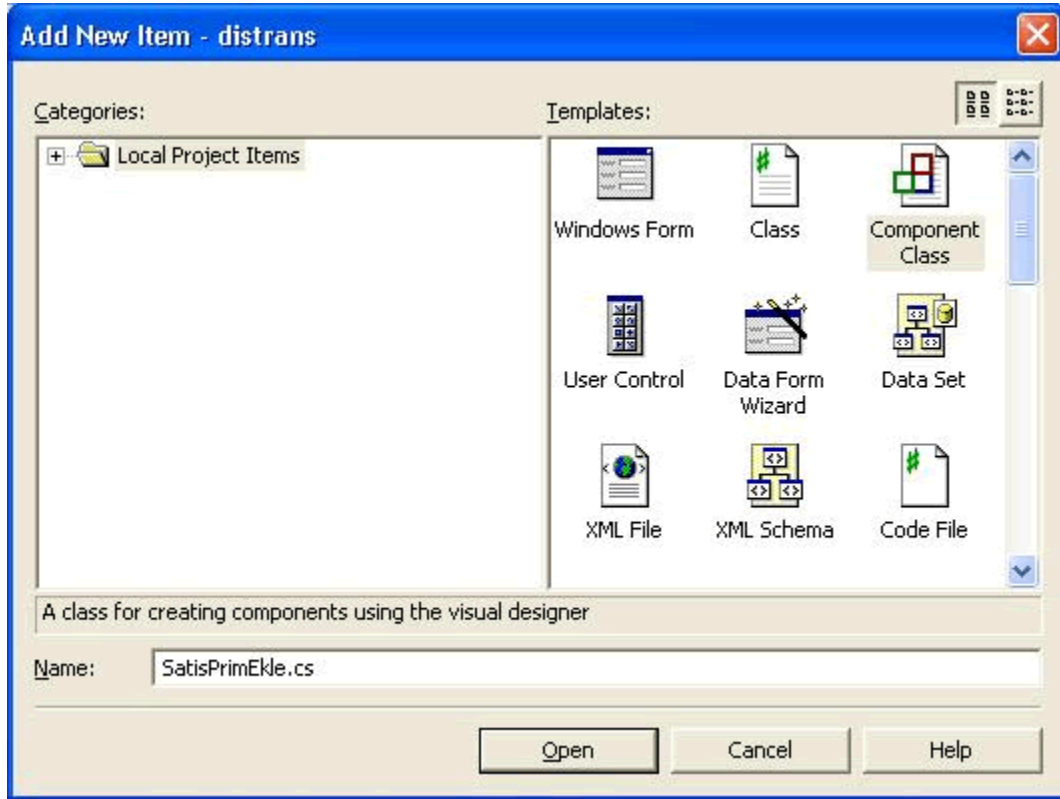


Şekil 3. Primler tablosunun yapısı.

Uygulamamızda Satışlar 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.

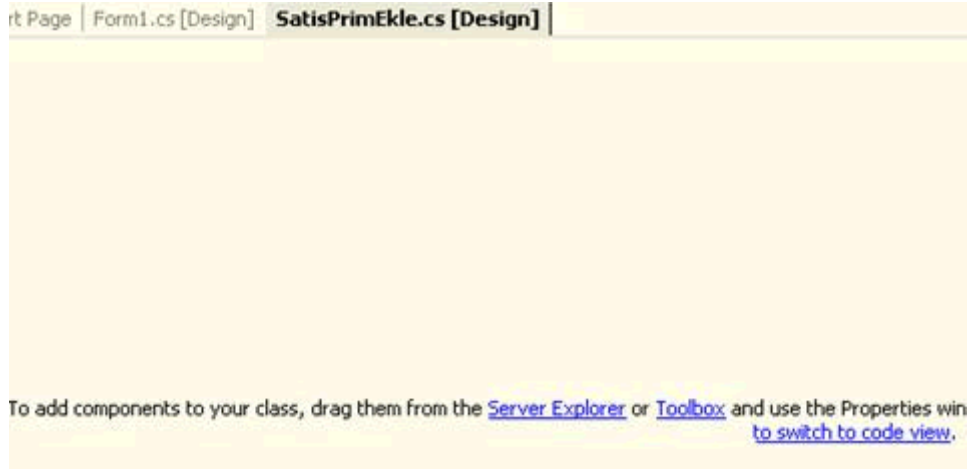
Şekil 4. Formun yapısı.

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.



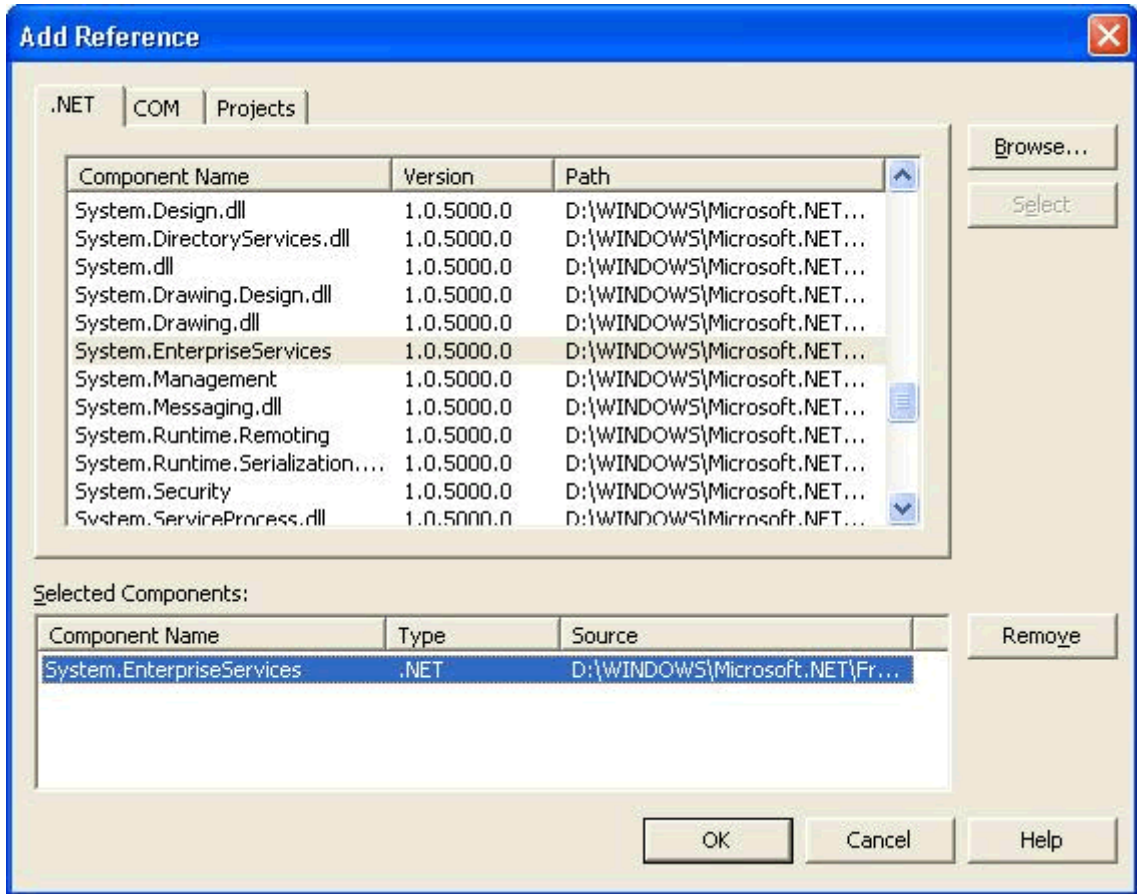
Şekil 5. Component Ekleme.

Ben componentimize SatisPrimEkle adini 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 içerisinde 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 değişimiz 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 distrans
{

```

/* [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 dır.

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. */

```

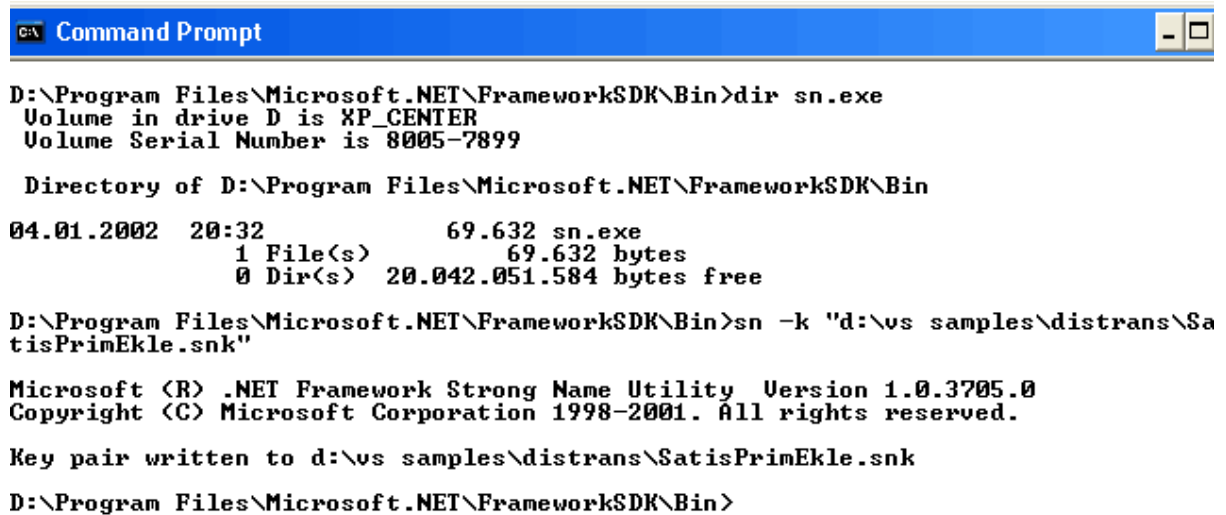
[AutoComplete()]public string VeriGonder(string ad,string soyad,double satisTutari)
{
    SqlConnection conFriends = new SqlConnection("initial catalog=Friends;data
source=127.0.0.1;integrated security=sspi;packet size=4096");
    SqlConnection conIstanbulMerkez = new SqlConnection("initial
catalog=IstanbulMerkez;data 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 veritabanlarında 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+ sevişi
transaction'ı otomatik olarak sonlandırır ve metodumuz geriye ISLEM TAMAM stringini
döndürür. */
    }
}
}
```

Component' imizi oluşturduktan sonar bunu Sunum Katmanındaki uygulamalarda kullanabilmek ve COM+ Servisi'ne de 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\Sa
tisPrimEkle.snk"

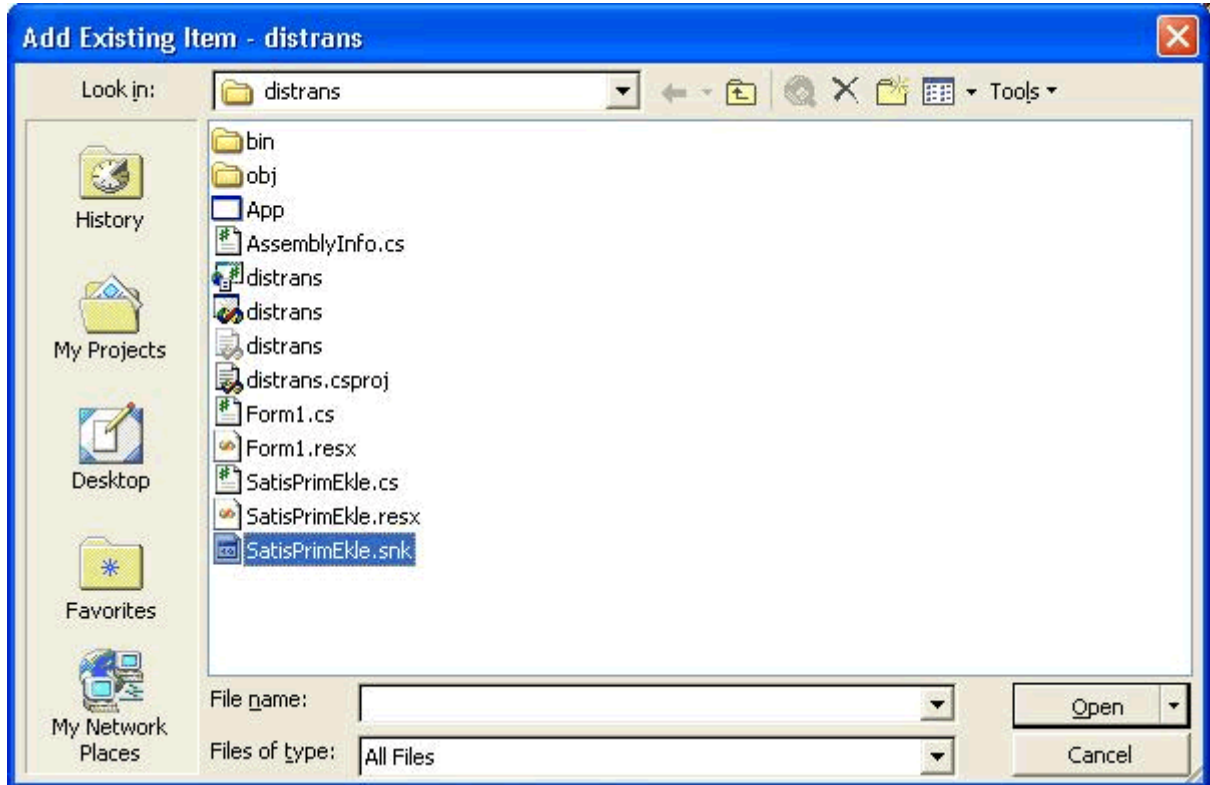
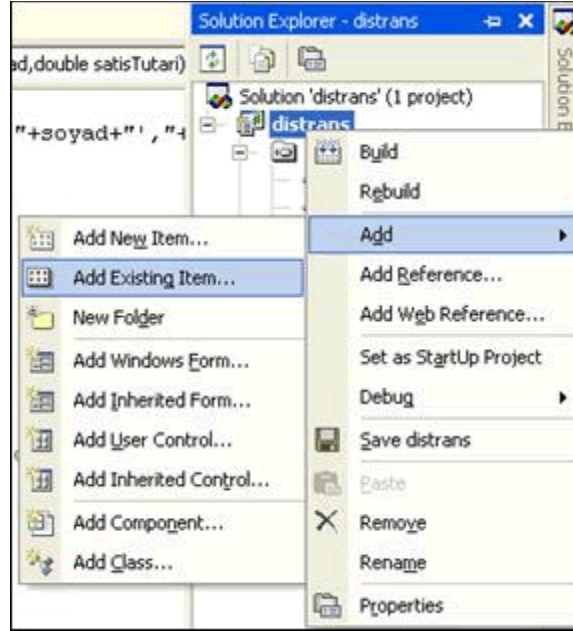
Microsoft (R) .NET Framework Strong Name Utility Version 1.0.3705.0
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.

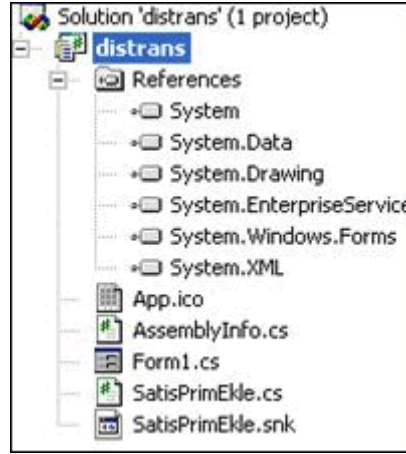
Key pair written to d:\vs samples\distrans\SatisPrimEkle.snk

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 Existing 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 kullanım.

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.

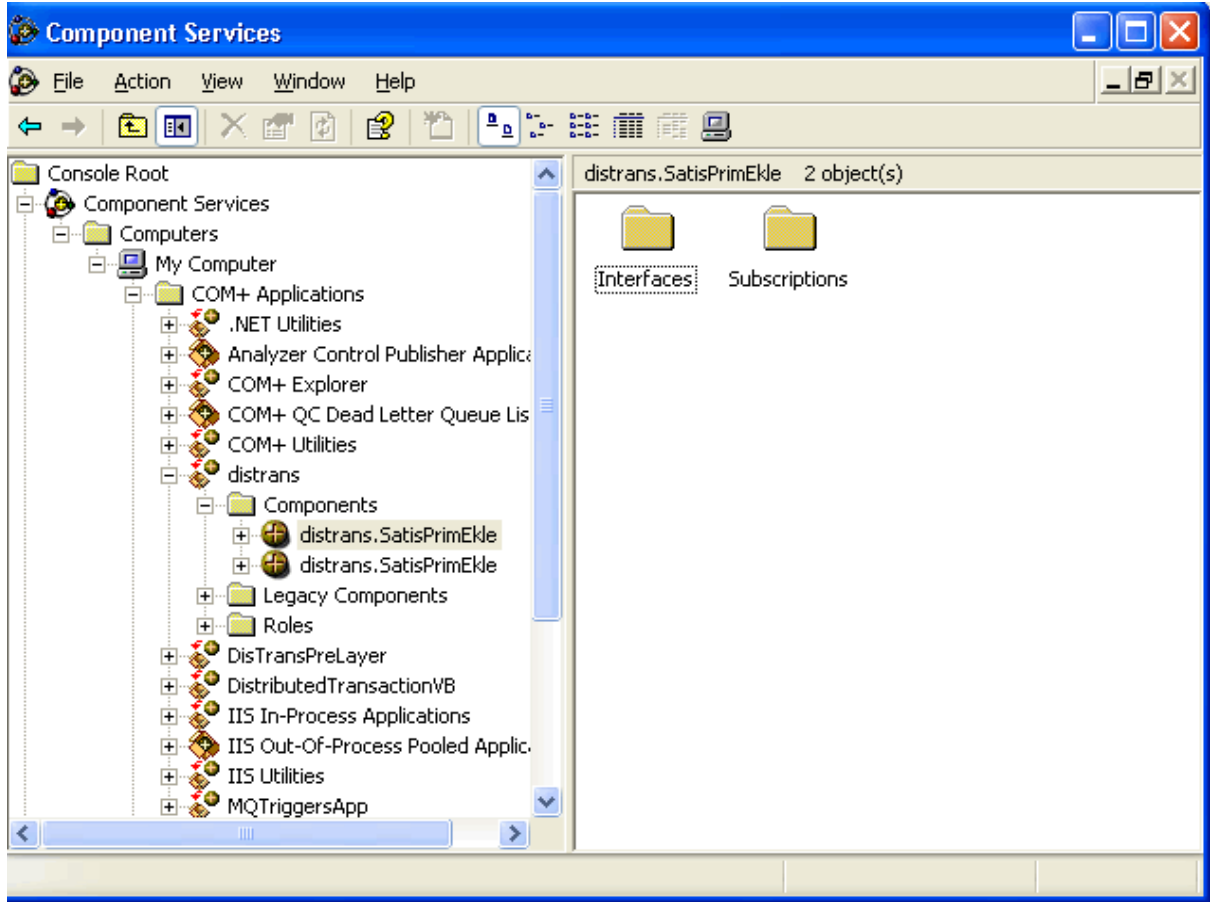
	ID	Ad	Soyad	Prim
1	1	Burak	Senyurt	150000000

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. Component 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 - 21 Kasım 2003 Cuma

windows forms,

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 çıkmar ve kendimize olan güvenimizi dahada arttırır.

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 içersinde bu bildirimi gerçekleştirmeliyiz. Ayrıca sınıfımızın Form nesnesine ait elemanlarıda kullanabilmesi için, Form sınıfından türetmeliyiz(Inherting). 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

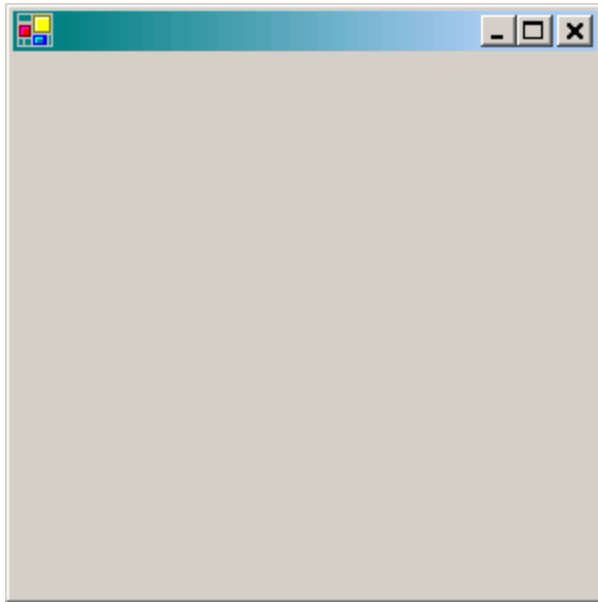
Directory of D:\vs samples\CreateForm

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());
    }
}
```

Bu yazım tekniği daha anlamlı değil mi ? Kodumuzu bu şekilde değiştirip çalıştırdığımızda 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
        Enumeration'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 oluşturuyor ve özelliklerini belirliyoruz */
    }
}

```

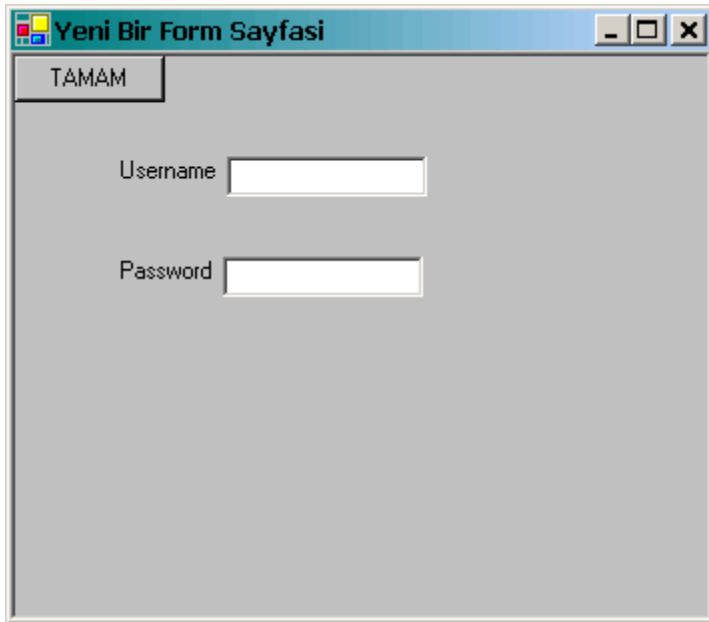
```

        btnOK=new Button();
        btnOK.Text="TAMAM";
        btnOK.Location=new Point(0,0);
        /* Buraya btnOK nesnesi için olay procedure tanımı eklenecek. */
        /* Şimdi 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 tasarını geliştiriyoruz.

Şimdi işin en önemli kısılarından 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 kullanırız.

```
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.EventHandler 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

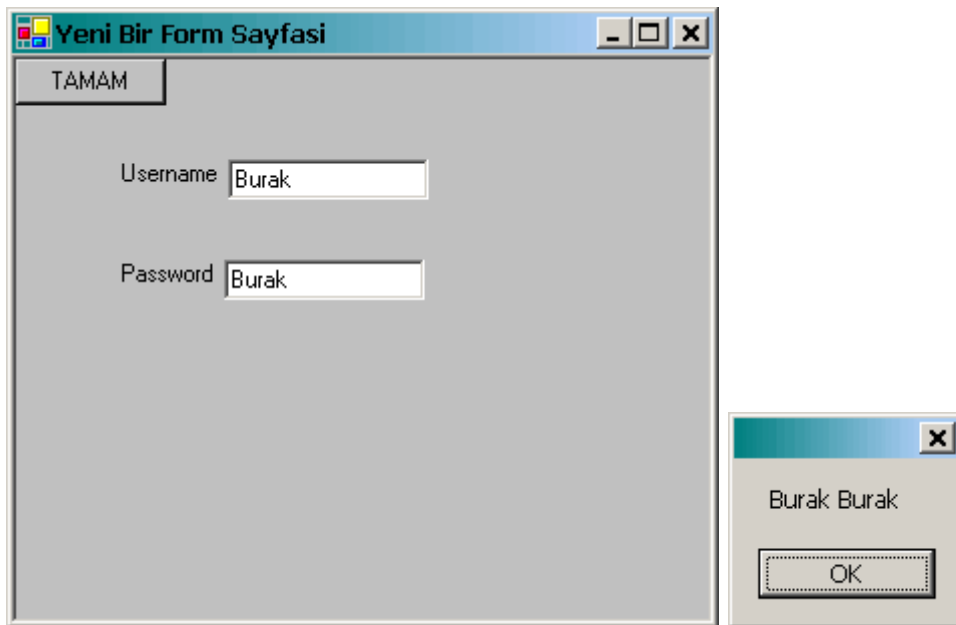
/ Buraya btnOK nesnesi için olay procedure tanımı eklenecek */* yazan yere , aşağıdaki kod satırını ekliyoruz.

```
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 - 30 Kasım 2003 Pazar

C#,

Değerli Okurlarım Merhabalar.

Bugünkü makalemizde, C# metodlarında önemli bir yere sahip olduğunu düşündüğüm params anahtar kelimesinin nasıl kullanıldığını incelemeye çalışacağız. Bildiğiniz gibi metotlara verileri parametre olarak aktarabiliyor ve bunları metod içersinde 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*/

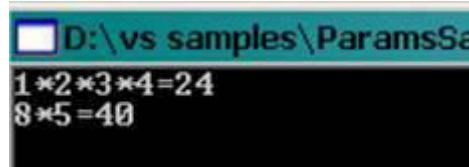
        public int Carpim(params int[] deger)
        {
            int sonuc=1;
            for(int i=0;i<deger.Length;++i) /*Metoda gönderilen elemanlar doğal
            olarak bir dizi oluştururlar. Bu dizideki elemanlara bir for döngüsü ile kolayca erişebiliriz.
            Dizinin eleman sayısını ise Length özelliği ile öğreniyoruz.*
            {
                sonuc*=deger[i];
                /* Burada metoda geçirilen integer değerlerin birbirleri ile
                çarpılmasını sağlıyoruz*/
            }
            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 deęişik 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 ę n , 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)
    {

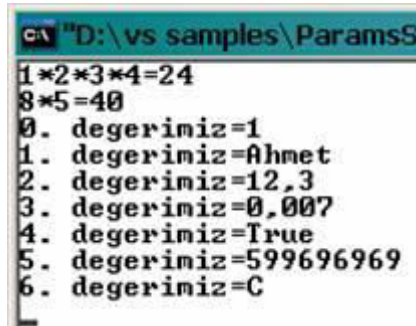
```

```

        Console.WriteLine("{0}. deęerimiz={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 deęişik tiplerde (int,Float,Decimal,bool, String) parametreler gönderiyoruz. İşte sonuç;



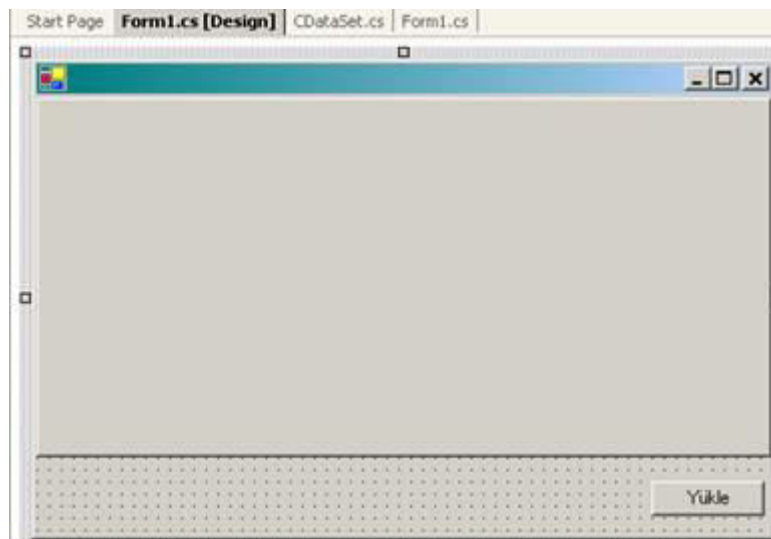
```

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

```

Şekil 2. params object[] kullanımı.

Şimdi dilerseniz daha işe yarar bir örnek üzerinde konuyu pekiştirmeye çalışalım. Örneğin deęişik 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 nesnelere yükleyecek. Kodları inceledikçe örneğimizi daha iyi anlayacaksınız.



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;

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerU	UnitPrice
▶	1	Chai	1	1	10 boxes x 20	18,0000
	2	Chang	1	1	24 - 12 oz bot	19,0000
	3	Aniseed Syru	1	2	12 - 550 ml b	10,0000
	4	Chef Anton's	2	2	48 - 6 oz jars	22,0000
	5	Chef Anton's	2	2	36 boxes	21,3500
	6	Grandma's B	3	2	12 - 8 oz jars	25,0000
	7	Uncle Bob's	3	7	12 - 1 lb pkgs	30,0000
	8	Northwoods	3	2	12 - 12 oz jar	40,0000
	9	Mishi Kobe Ni	4	6	18 - 500 g pk	97,0000

Yükle

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.

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerU	UnitPrice
▶	1	Chai	1	1	10 boxes x 20	18,0000
	2	Chang	1	1	24 - 12 oz bot	19,0000
	3	Aniseed Syru	1	2	12 - 550 ml b	10,0000
	4	Chef Anton's	2	2	48 - 6 oz jars	22,0000
	5	Chef Anton's	2	2	36 boxes	21,3500
	6	Grandma's B	3	2	12 - 8 oz jars	25,0000
	7	Uncle Bob's	3	7	12 - 1 lb pkgs	30,0000
	8	Northwoods	3	2	12 - 12 oz jar	40,0000
	9	Mishi Kobe Ni	4	6	18 - 500 g pk	97,0000

Yükle

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 - 30 Kasım 2003 Pazar

ado.net, dataset, xml, writexml,

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
dokümanına yazıyoruz. */

        conFriends.Close(); // Bağlantımızla işimiz bittiğinden kapatıyoruz.
    }
}

```

Bu uygulamayı çalıştırdığımızda, D:\Kitaplar.xml isimli bir dosyanın oluştuğunu görürüz. Bu dosyayı açtığımızda ise aşağıda yer alan xml kodlarını elde ederiz. Gördüğünüz 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.

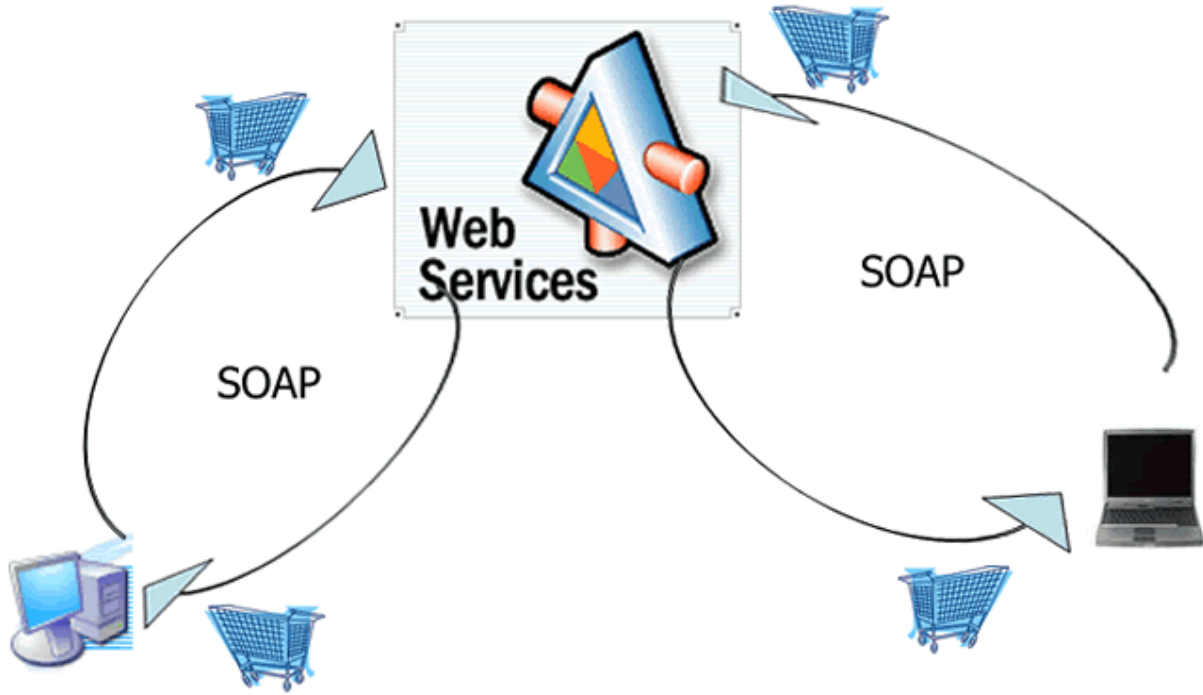
Basit Bir Web Service Uygulaması - 30 Kasım 2003 Pazar

xml web services,

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 zihninizde daha kolay canlandırabiliriz .



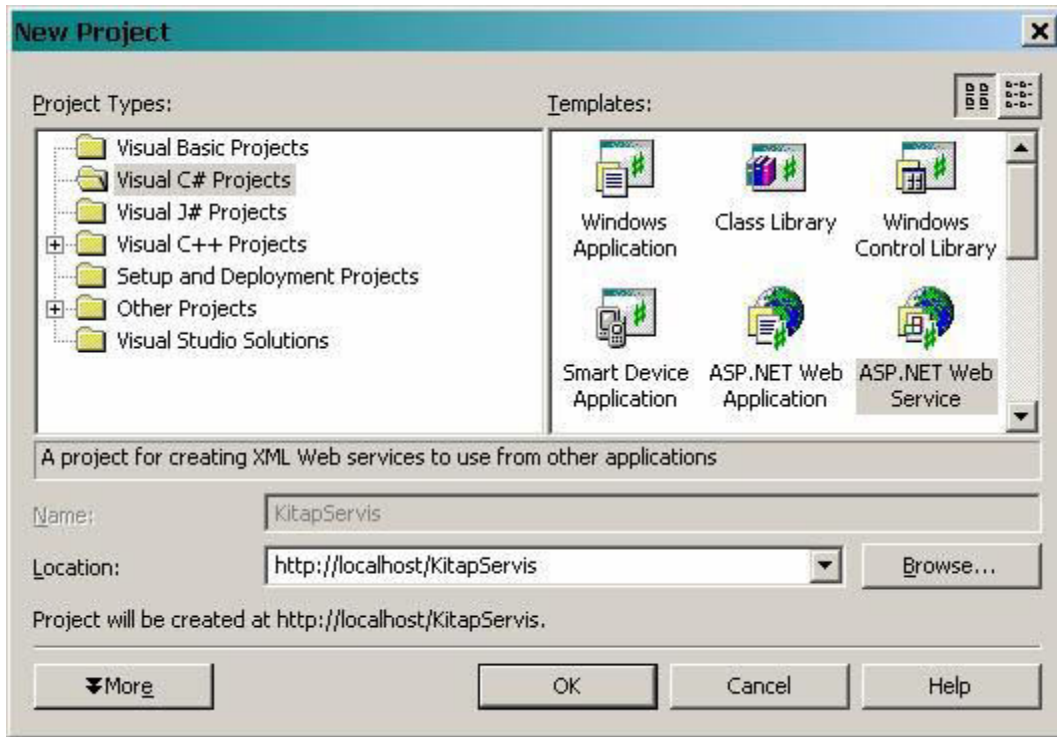
Şekil 1 . Web Servislerinin yapısı

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 .

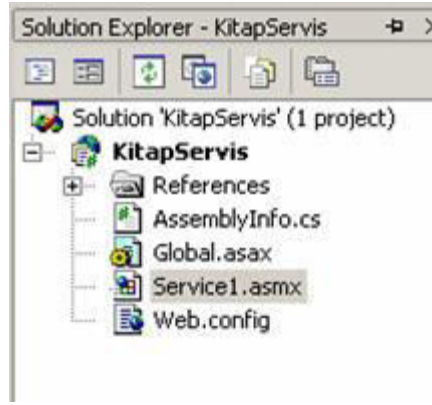
Şimdi, basit bir web servis uygulaması geliştireceğiz. Öncelikle web servis'imizi yazacağız. Daha sonra ise, bu web servisini kullanacağımız bir windows application geliştireceğiz. Normalde birde web proxy uygulaması geliştirmemiz gerekiyor. Ancak bu işi VS.NET'e bırakacağız.

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 oluştuğunu görürüz. Web servislerinin dosya uzantısı asmx dir. Bu uzantı çalışma zamanında veya bir tarayıcı penceresinde bu dosyanın bir web servis olduğunu belirtir.



Şekil 3. Web servisleri asmx uzantısına sahiptir.

İTo 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.

```

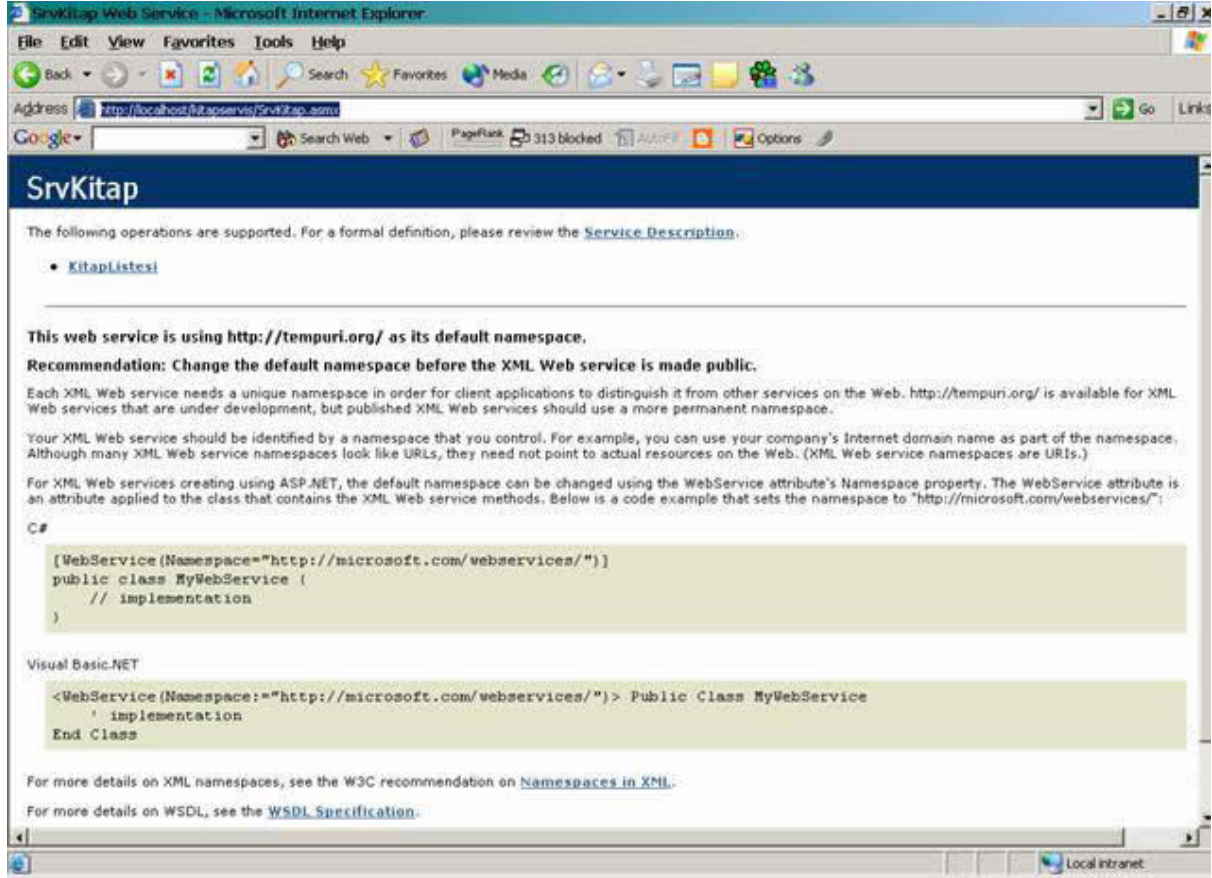
[WebMethod]
public DataSet KitapListesi()
{
    SqlConnection conFriends=new SqlConnection("data source=localhost;initial
catalog=Friends;integrated security=sspi");
    SqlDataAdapter da=new SqlDataAdapter("Select Adi,Fiyat From Kitaplar",conFriends);
    DataSet ds=new DataSet();
    da.Fill(ds);
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

bu durumda, aşağıdaki ekran görüntüsünü alırız.



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:lsDataSet="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>
      <Table diffgr:id="Table6" msdata:rowOrder="5">
```



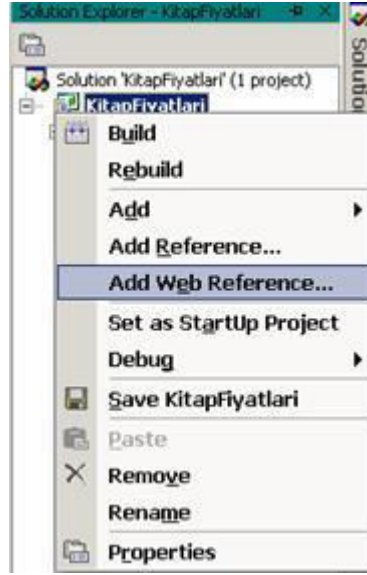
```
<Adi>Microsoft Visual Basic 6 Temel Kullanım Kılavuzu Herkes İçin!</Adi>
<Fiyat>15.0000</Fiyat>
</Table>
<Table diffgr:id="Table7" msdata:rowOrder="6">
  <Adi>ASP ile E-Ticaret Programcılığı</Adi>
  <Fiyat>25.0000</Fiyat>
</Table>
<Table diffgr:id="Table8" msdata:rowOrder="7">
  <Adi>ASP 3.0 Active 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.



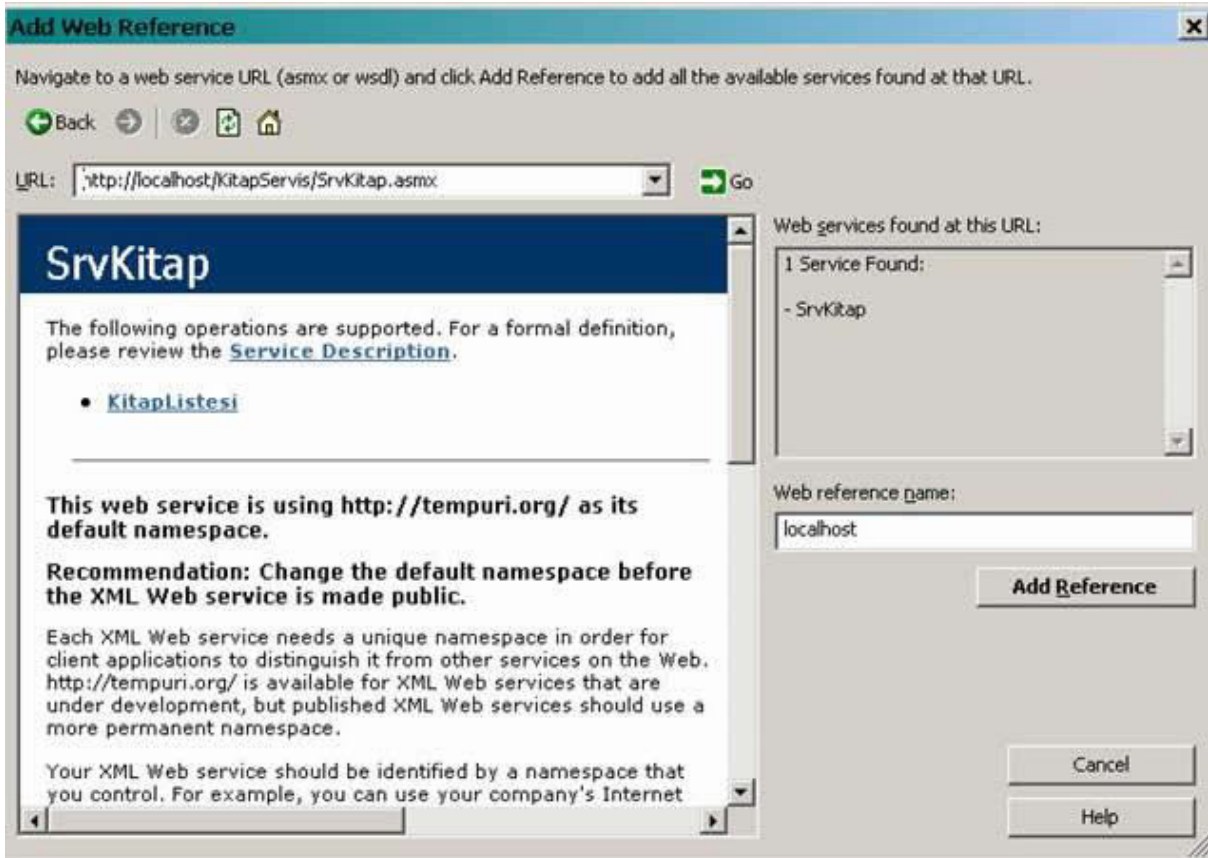
Şekil 6 Form Tasarımı.

Formumuz bir adet datagrid nesnesi ve bir adet button nesnesi içeriyor. Button nesnemize tıkladığımızda, web servisimizdeki KitapListesi adlı metodumuzu çağırarak ve dataGridView'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.



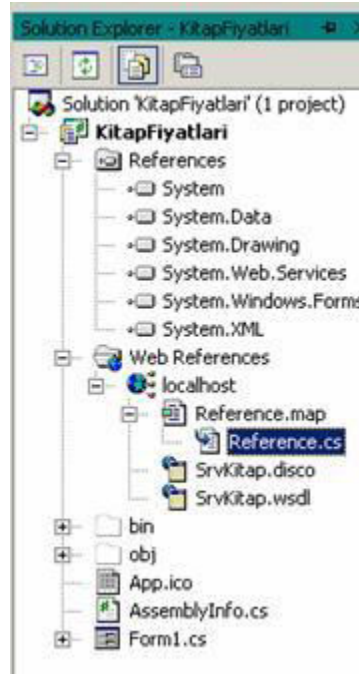
Şekil 7. Uygulamaya bir Web Reference eklemek.

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 içerdği 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 KitapFiyatları.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 işyeceğim. 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)
{

```

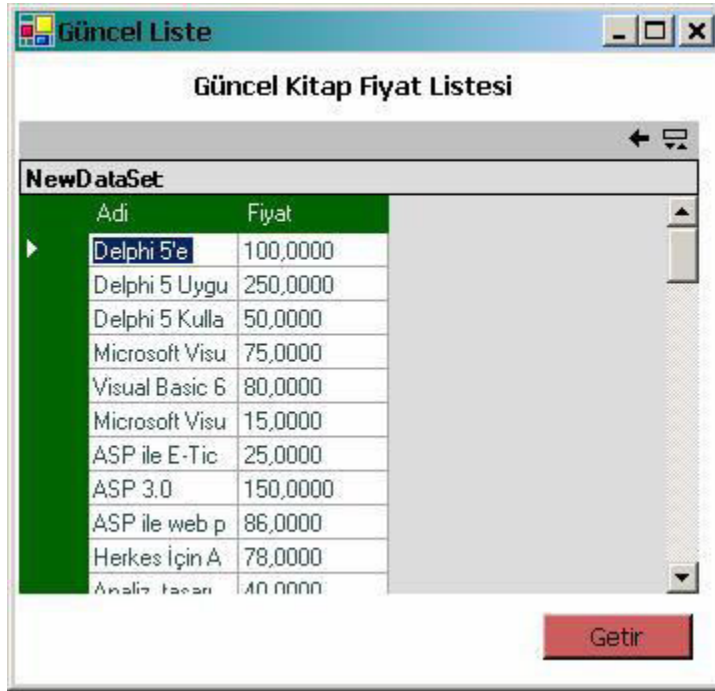
```
localhost.SrvKitap srv=new localhost.SrvKitap()); /* Servisimizi kullanabilmek için bu servisten bir örnek nesne yaratıyoruz*/
```

```
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.

Enumerators - 01 Aralık 2003 Pazartesi

C#, enums, enumerator, type, class, delegate, interface, cts,

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 anlaşılabilmelerini sağlamaktır. Örneklerimizde bu konuyu çok daha iyi anlayacaksınız. Bir Numaralandırıcı tanımlamak için aşağıdaki syntax kullanılır.

Kapsam belirteçleri

```
enum numaralandiriciAdi
{
    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);
    }
}

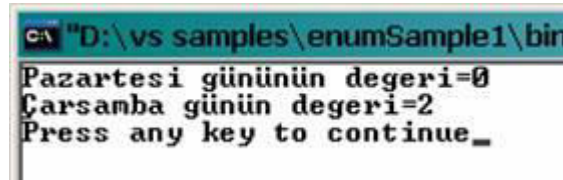
```

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
}

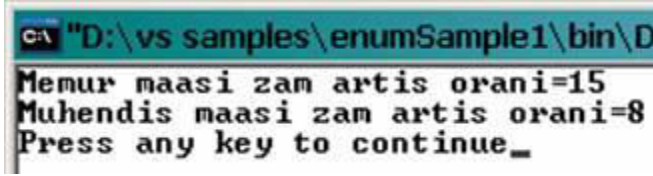
static void Main(string[] args)
{

```

```

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 değer türünde belirtmemiz gerekmektedir.Örneğin,

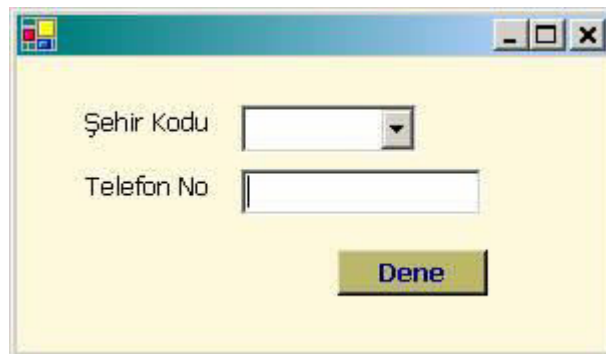
```

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 atanabilmiştir. 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.



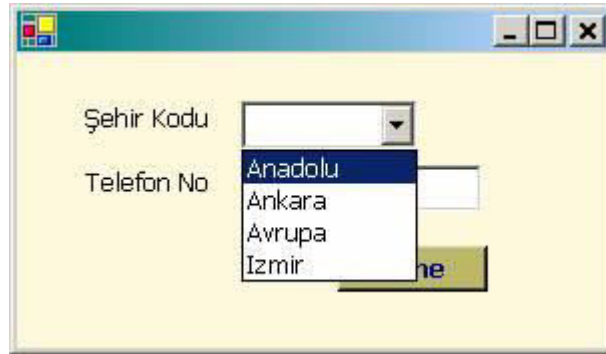
Şekil 4. Form Tasarımımız.

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.

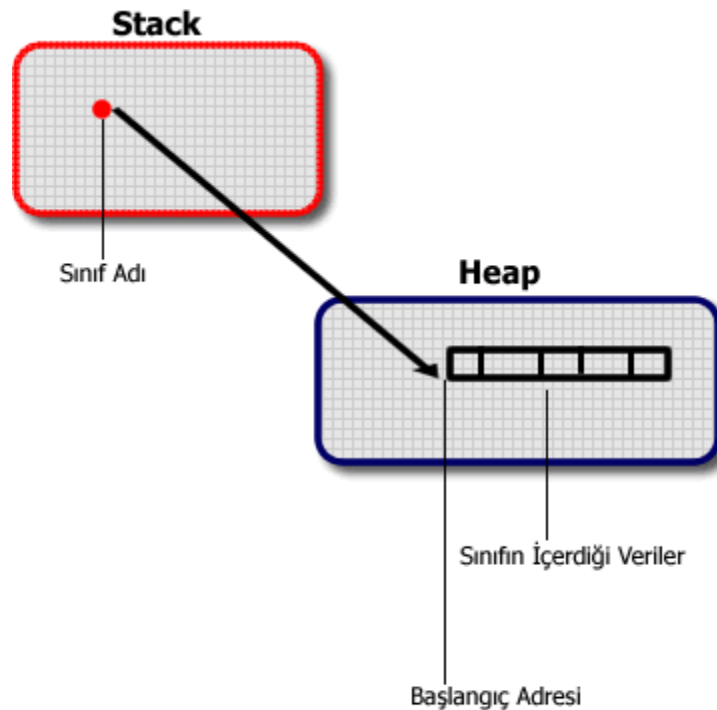
Struct Kavramı ve Class ile Struct Arasındaki Farklar - 04 Aralık 2003 Perşembe

struct, class, c#, c# temelleri,

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 benzerlik gösterirler. Sınıf gibi tanımlanırlar. Hatta sınıflar gibi, özellikler, metodlar, veriler, yapıcılar vb... içerebilirler. Buna karşın sınıflar ile yapılar arasında çok önemli farklılıklar vardır.

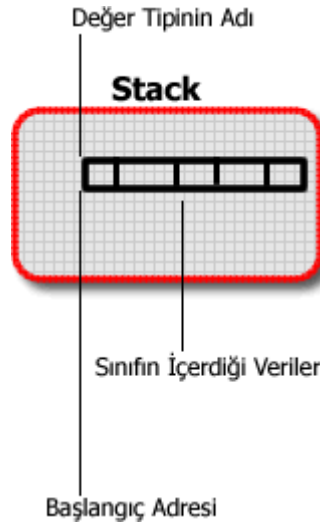
Herşeyden önce en önemli fark, yapıların değer türü olması ve sınıfların referans türü olmasıdır. Sınıflar referans türünden oldukları için, bellekte tutuluş biçimleri değer türlerine göre daha farklıdır. Referans tiplerinin sahip olduğu veriler belleğin öbek(heap) adı verilen tarafında tutulurken, referansın 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.



Referans Tiplerinin Bellekte Tutuluş Şekli

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 kez 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 tanımlamış 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 tanımlamış olduğumuz alanlara mutlaka ilk değerlerini vermemiz gerekir. Ancak yine de 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.

```
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());
    }
}

```

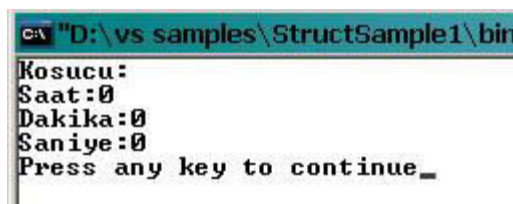
Yukarıdaki kod derlenmeyecektir. Nitekim derleyici **“Use of unassigned local variable 'z'”** hatası ile z yapısı için ilk değerlerin atanmadığını bize söyleyecektir. Ancak z isimli Zaman yapı türünü new anahtarı ile tanımlarsak durum değişir.

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.



```

C:\> "D:\vs samples\StructSample1\bin
Kosucu:
Saat:0
Dakika:0
Saniye:0
Press any key to continue_

```

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 farklılıklarıda 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;
    }
}

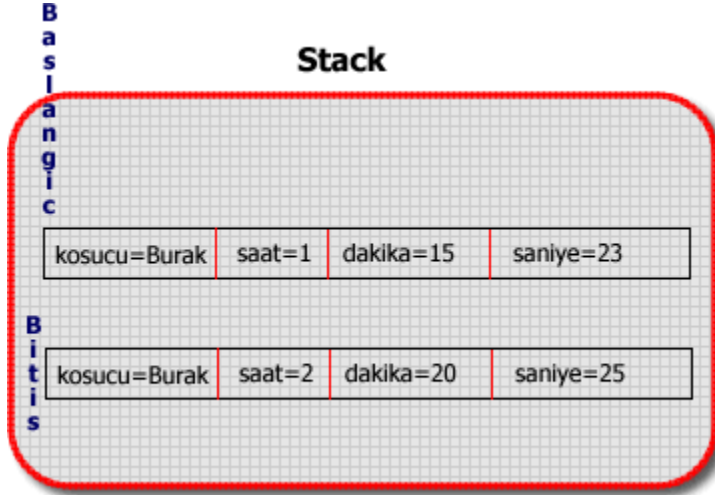
public int Dakika
{
    get
    {
        return dakika;
    }
    set
    {
        dakika=value;
    }
}

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.

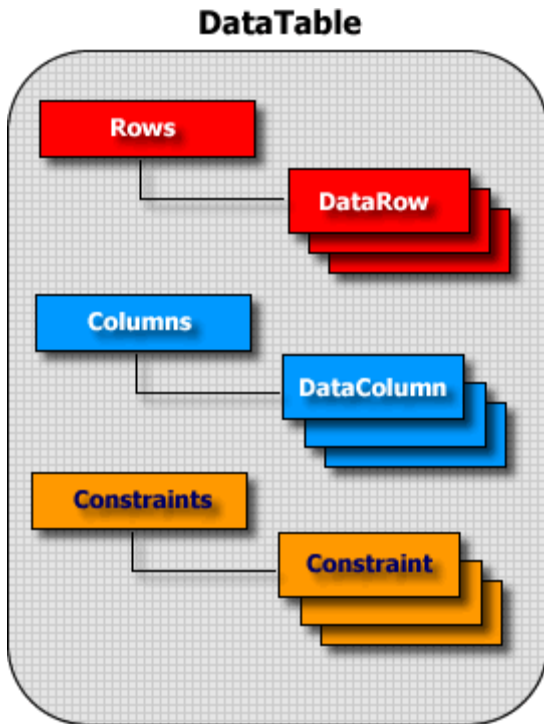
DataTable Sınıfını Kullanarak Programatik Olarak Tablolar Oluşturmak-1 - 04 Aralık 2003 Perşembe

ado.net, datatable,

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.

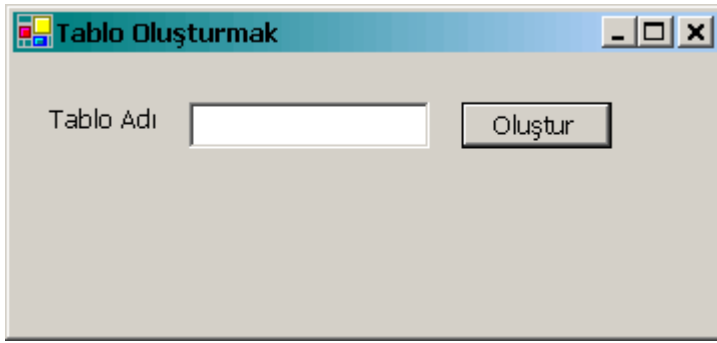


Şekil 1 DataTable mimarisi

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 üzölmeyin 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.

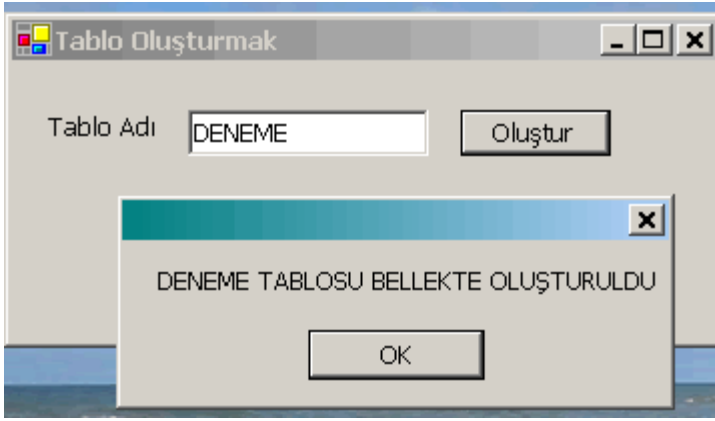


Şekil 2. Formun ilk hali

Ve kodlar,

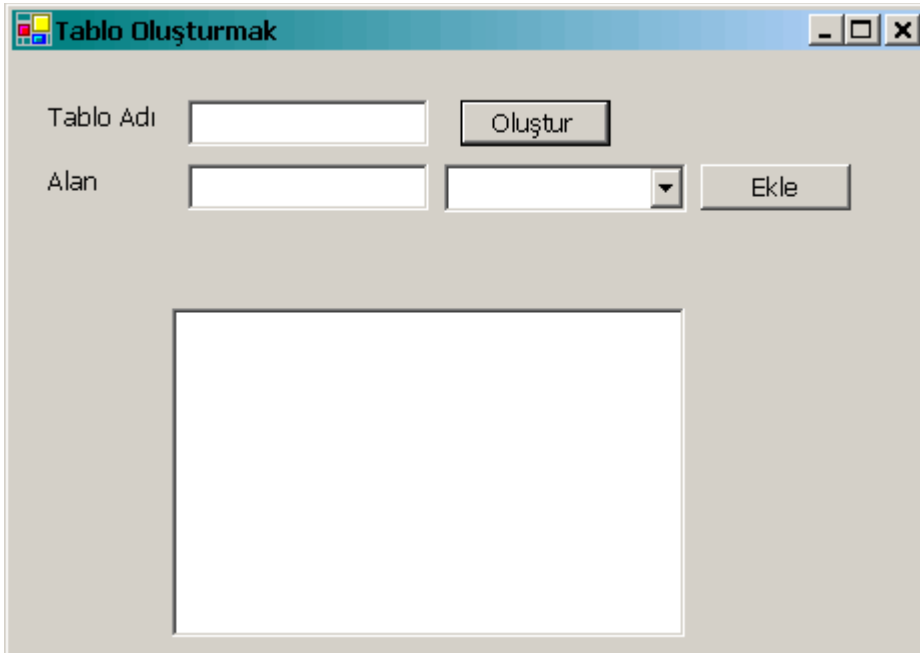
```
private void btnTabloOlustur_Click(object sender, System.EventArgs e)
{
    /* Bir tabloyu bellekte temsil edecek bir datatable nesnesi oluşturarak işe başlıyoruz.
    Tablomuz txtTabloAdi isimli TextBox'a girilene değeri isim olarak veriyoruz */
    DataTable dt=new DataTable(txtTabloAdi.Text);
    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 önünce 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.

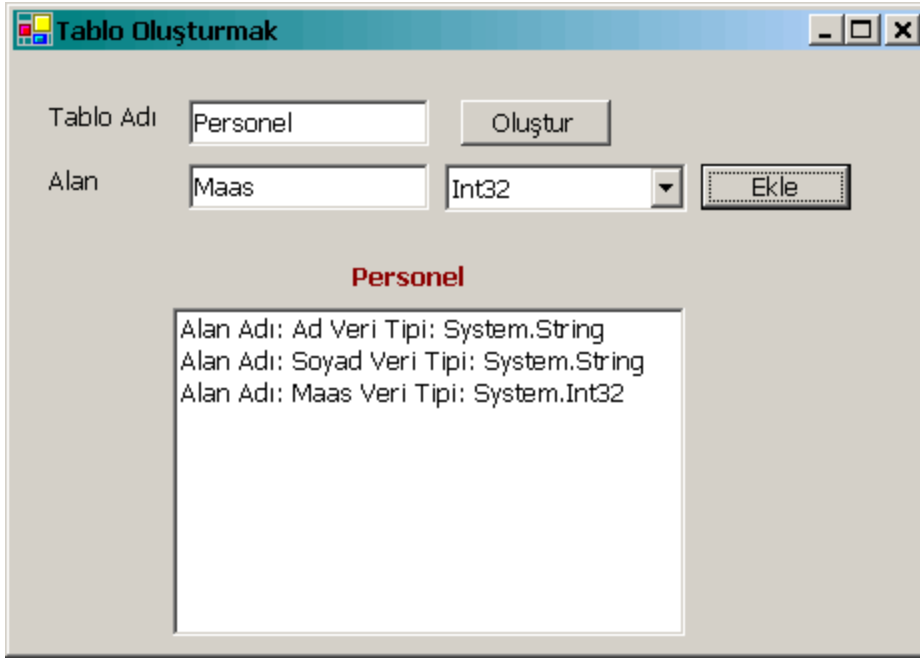
DataTable dt; /* DataTable nesnemizi uygulama boyunca kullanabilmek için tüm metodların dışında tanımladık. */

```
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 ilede bu alanın
    veri türünü öğreniyoruz.*/
}
```

Uygulamamızı deneyelim.



Şekil 5. Alanlarımızı da 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 alacak 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"); /* 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; /* Alanın değerleri 1'er artacak.*/
dt.Columns.Add(dcID); /* Yukarıda özellikleri belirtilen ID alanı tablomuza ekleniyor. */
/* Aşağıdaki kod satırları ile ID isimli alanı Primary 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 sonraki makalemizde oluşturduğumuz bu tabloya nasıl veri ekleneceğini göreceğimiz çok kısa bir uygulama yazacağız. Hepinizi mutlu günler dilerim.

DataTable Sınıfını Kullanarak Programatik Olarak Tablolar Oluşturmak- 2 - 05 Aralık 2003 Cuma

ado.net, datatable,

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 DataGrid 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 nesnesinde 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 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"); /* 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; /* Alanın değerleri 1'er artacak.*/

dt.Columns.Add(dcID); /* Yukarıda özellikleri belirtilen ID alanı tablomuzaya ekleniyor. */

/* Aşağıdaki kod satırları ile ID isimli alanı Primary 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");

}

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.*/

}

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 oluşturulup formumuza ekleniyor. Top
    özelliğinin ayarlanışına dikkatinizi çekmek isterim. 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 btnKayıtEkle_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 gelmesin. 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 dataGridView 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,

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 - 06 Aralık 2003 Cumartesi

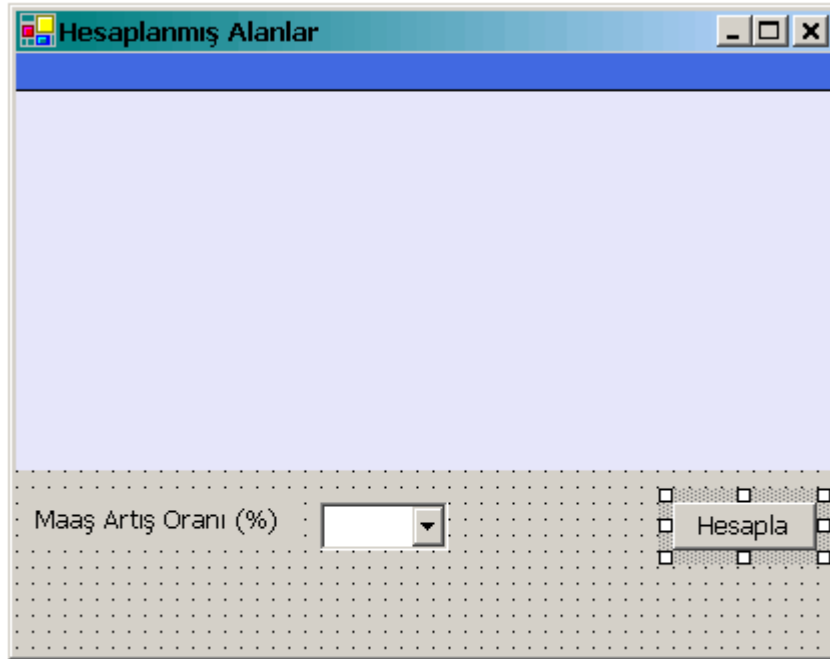
ado.net, data column, expression,

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 gerçekleştireceğiz.



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
hesaplama, 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 oranında 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 (%)

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 (%)

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 kaybınada 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 sizin de 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 - 09 Aralık 2003 Salı

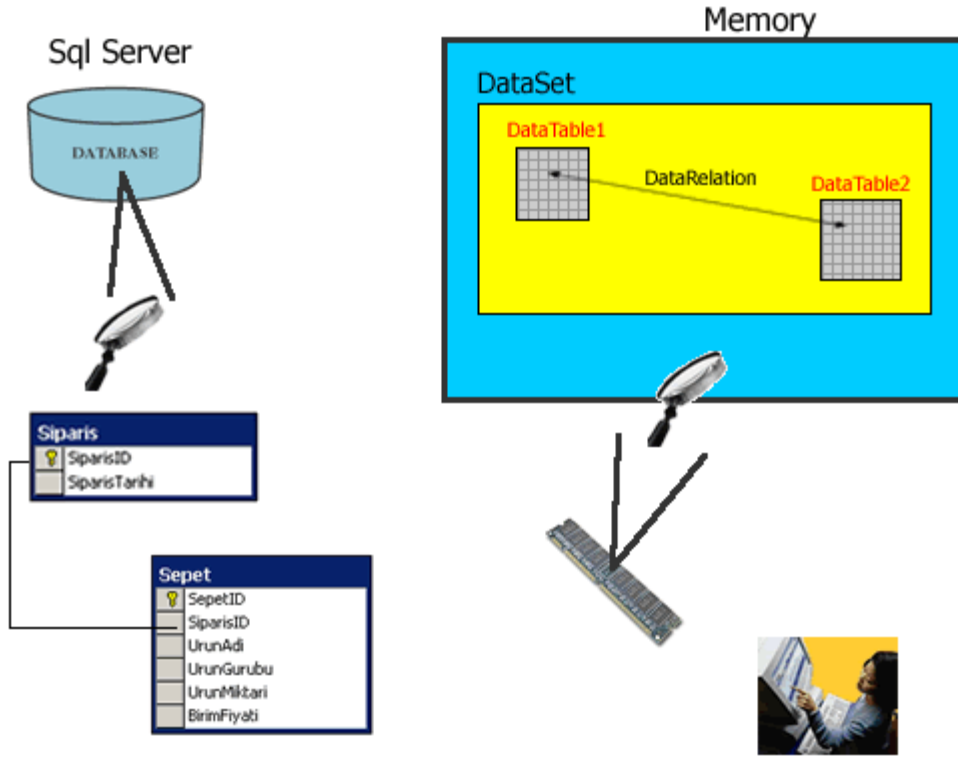
ado.net, dataset,

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 sınıftır. 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

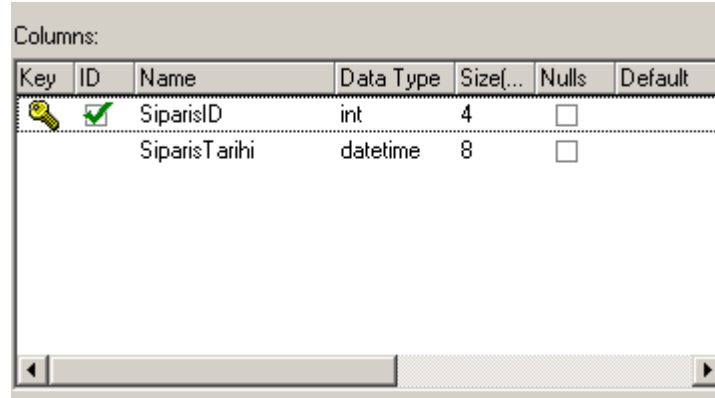
Bir DataRelation nesnesi oluşturmak için kullanabileceğimiz Constructor metodlar şunlardır.

- 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 ait 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 tarzaları 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.

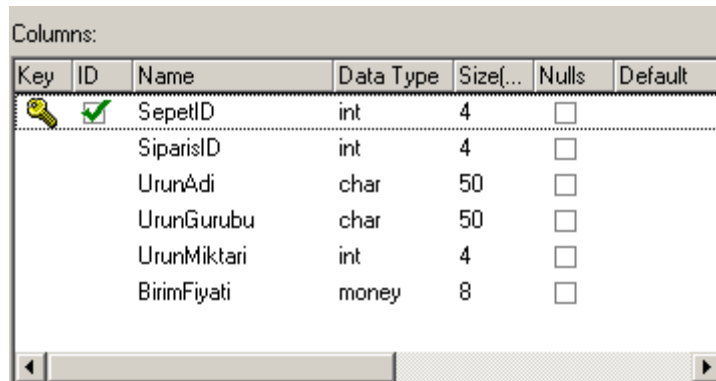
Şimdi gelin kısa bir uygulama ile bu konuyu işleyelim. Uygulamamızda kullanılan tablolara ait alanlar ve özellikleri şöyledir. İlk tablomuz Siparis isimli tablomuz. Bu tabloda kullanıcının vermiş olduğu siparişin numarası ve tarihi ile ilgili bilgiler tutuluyor. Bu tablo bizim parent(master) tablomuzdur.



Key	ID	Name	Data Type	Size(...)	Nulls	Default
		SiparisID	int	4	<input type="checkbox"/>	
		SiparisTarihi	datetime	8	<input type="checkbox"/>	

Şekil 2. Siparis Tablosu

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



Key	ID	Name	Data Type	Size(...)	Nulls	Default
		SepetID	int	4	<input type="checkbox"/>	
		SiparisID	int	4	<input type="checkbox"/>	
		UrunAdi	char	50	<input type="checkbox"/>	
		UrunGurubu	char	50	<input type="checkbox"/>	
		UrunMiktari	int	4	<input type="checkbox"/>	
		BirimFiyati	money	8	<input type="checkbox"/>	

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. */

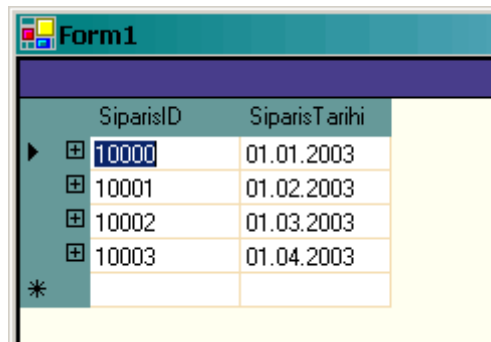
DataRelation drSiparisToSepet=new
DataRelation("Siparis_To_Sepet",dtSiparis.Columns["SiparisID"],dtSepet.Columns["SiparisID"]);
/* Artık oluşturduğumuz bu DataTable nesnelerini ve DataRelation nesnemizi DataSet
nesnemize ekleyebiliriz. Dikkat edecek olursanız, DataRelation nesnemizi dataSet
nesnemizin Relations koleksiyonuna ekledik. DataRelation nesneleri DataTable nesneleri
gibi DataSet'e ait ilgili koleksiyonlarda tutulmaktadır. 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
Siparis To Sepet	
10001	01.02.2003
Siparis To Sepet	
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)

Form1						
Siparisler: SiparisID: 10001 SiparisTarihi: 01.02.2003						
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 - 10 Aralık 2003 Çarşamba

ado.net,

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 tanımlası ile de bir alanın değerlerinin asla tekrar edilemeyeceği şartını bildirmiş oluruz. Bir yabancı anahtar kısıtlaması için ForeignKeyConstraint sınıfı kullanılır. Aynı şekilde bir tekillik 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. Sözelimi parent tabloda bir satırın silinmesi ile ilişkili tablodaki ilişkili satırlarında tümü silinir. None değeri verildiğinde tahmin edeceğiniz 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 özellikde 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 gerektiğide 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 başlatacak. 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	İsletim 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	5000000,0000	
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 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.

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 - 12 Aralık 2003

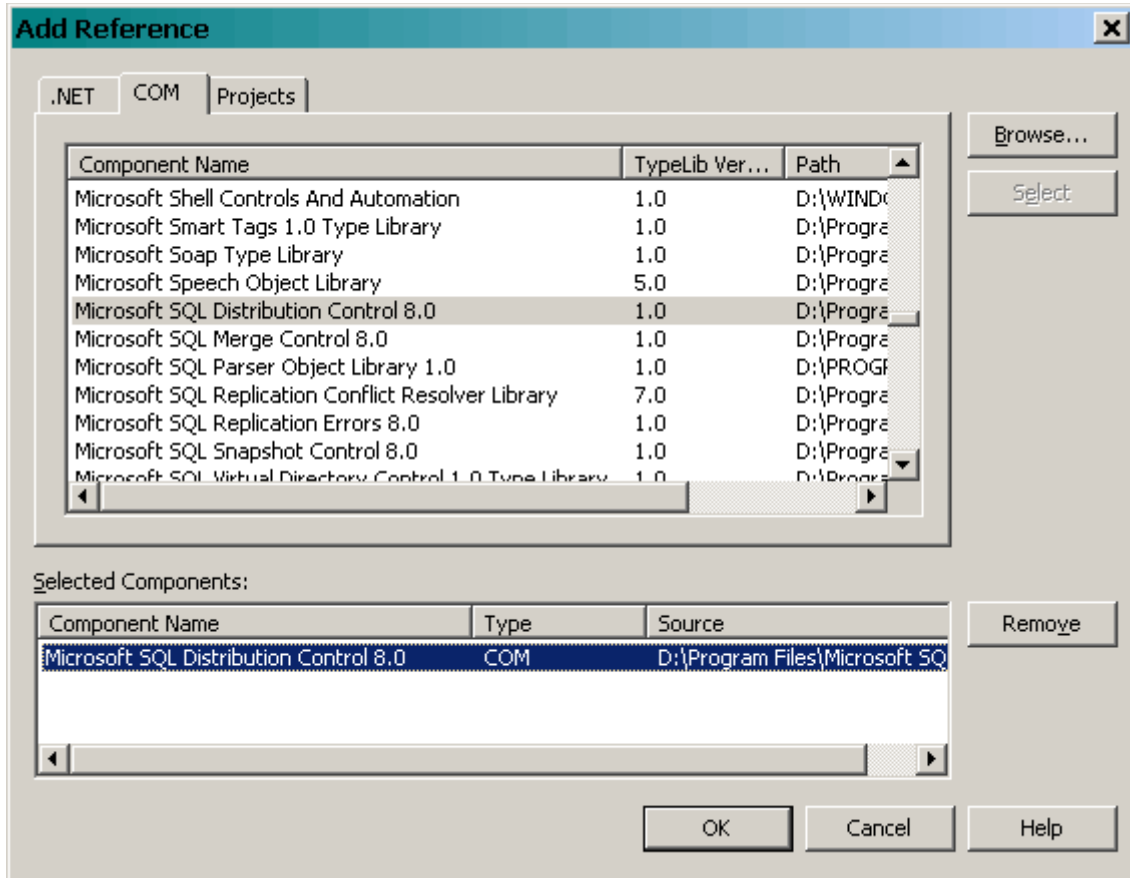
Cuma

ado.net, sql, dmo,

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 veritabanlarını 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 Sunucunuza 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 edeceğiniz 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 sınıfında yer alan SQLServerClass sınıfının
    bir örneği olacaktır. */
```

```
    srv.LoginSecure=false; /* Bu özellik true olarak belirlendiğinde, Sql Sunucusuna Windows
    Authentication şeklinde bağlanılır. Eğer false değerini verirse bu durumda Sql Server
    Authentication geçerli olur. İşte bu durumda SQLServerClass nesnesini Connect metodu ile
    Sql Sunucusuna bağlanırken geçerli bir kullanıcı adı ve şifre girmemiz gerekmektedir. */
```

```
    try
    {
        srv.Connect("BURKI","sa","CucP??80."); /* Bağlantı kurmak için kullandığımız Connect
        metodu üç parametre almaktadır. İlk parametre sql sunucusunun adıdır. İkinci parametre
        kullanıcı adı ve üçüncü parametrede şifresidir. Eğer LoginSecure=true olarak ayarlasaydık,
        kullanıcı adını ve şifreyi boş bırakıyorduk, nitekim Windows Authentication (windows
        doğrulaması) söz konusu olucaktı. */
```

```
        durumCubugu.Text="Sunucuya bağlanıldı..." +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, veritabanlarını
            temsil eden sınıf DatabaseClass sınıfıdır. */
            db.Name=this.txtVeritabaniAdi.Text; /* Veritabani nesnemizin name özelliği ile
            veritabanının adı belirlenir. */
            srv.Databases.Add(db); /* oluşturulan DatabaseClass nesnesi SQLServerClass
            sınıfının Databases koleksiyonuna eklenerek Sql Sunucusu üzerinde oluşturulması
            sağlanıyor. */
            durumCubugu.Text=db.Name.ToString()+" veritabani "+srv.Name.ToString()+" SQL
            Sunucusunda oluşturuldu";
        }
        catch(Exception hata)
        {
            MessageBox.Show(hata.Message);
        }
    }
    private void btnTabloOlustur_Click(object sender, System.EventArgs e)
    {
        try
        {
            tbl=new SQLDMO.TableClass(); /* Yeni bir tablo oluşturabilmek için SQL-DMO
            kütüphanesinde yer alan, TableClass sınıfı kullanılır. */
            tbl.Name=txtTabloAdi.Text; /* Tablomuzun ismini name özelliği ile belirliyoruz. */
            SQLDMO.ColumnClass dc; /* Tabloya eklenecek alanların her birisi birer
            ColumnClass sınıfı nesnesidir. */
            dc=new SQLDMO.ColumnClass(); /* Bir ColumnClass nesnesi yaratılıyor ve bu
            nesnenin gerekli özellikleri belirleniyor. Name özelliği 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 olusturulan TableClass nesnesi veritabanimizin tables koleksiyonuna
ekleniyor. Böylece Sql sunucusunda yer alan veritabani içinde olusturulmasi saglanmis
oluyor. */
db.Tables.Add(tbl);
durumCubugu.Text=tbl.Name.ToString()+" olusturuldu...";
}
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 veritabanlarinin sayisini
    aktariyoruz. */
    this.lstDatabases.Items.Add("Sunucudaki veritabani sayisi="+srv.Databases.Count);
    /* Simdi bir for döngüsü ile, srv isimli SQLServerClass nesnemizin Databases
    koleksiyonunda geziniyoru ve her bir databaseClass nesnesinin adini alip listBox nesnemize
    aktariyoruz. Burada index degerinin 1 den basladigina sifirdan baslamadigina 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 veritabani 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());
    }
}
}

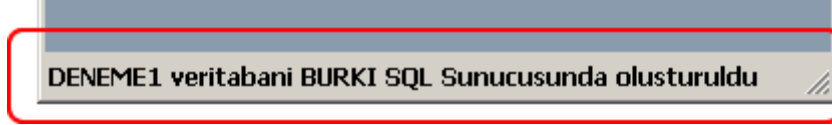
```

Şimdi uygulamamızı çalıştıralım ve öncelikle Sql Sunucumuza bağlanalım.



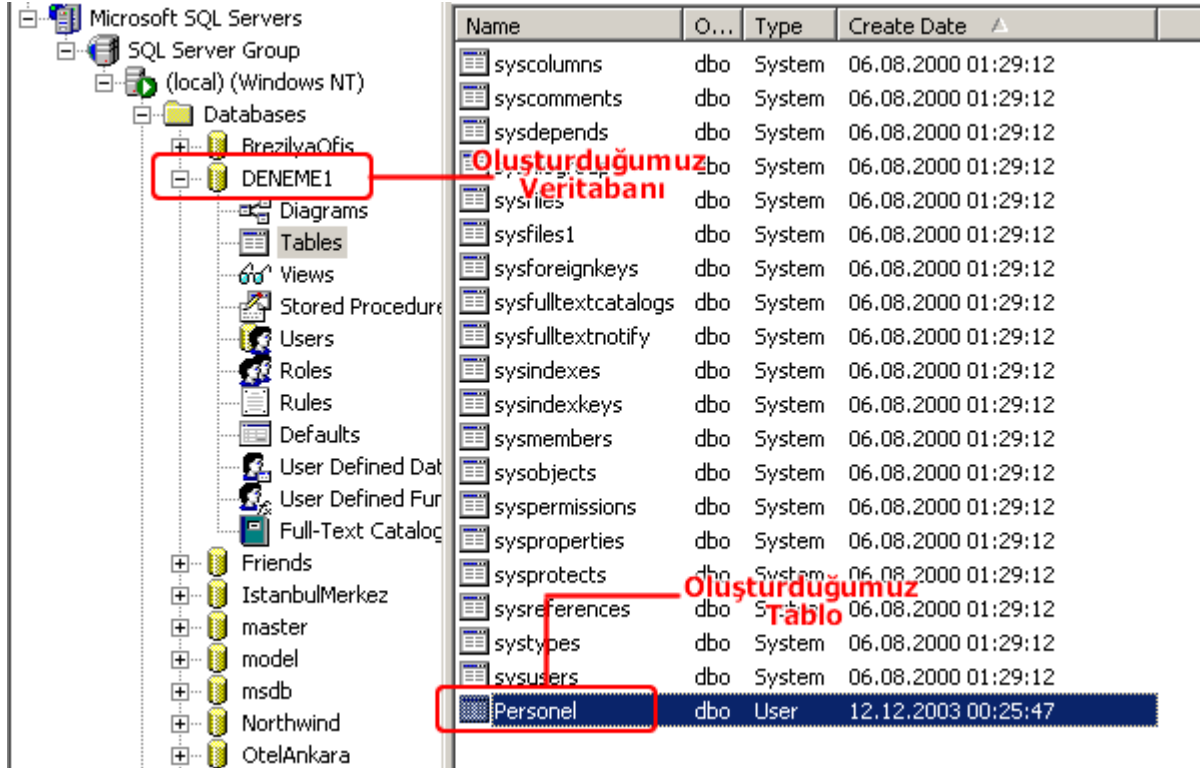
Şekil 3. Sunucumuza Bağlandık.

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



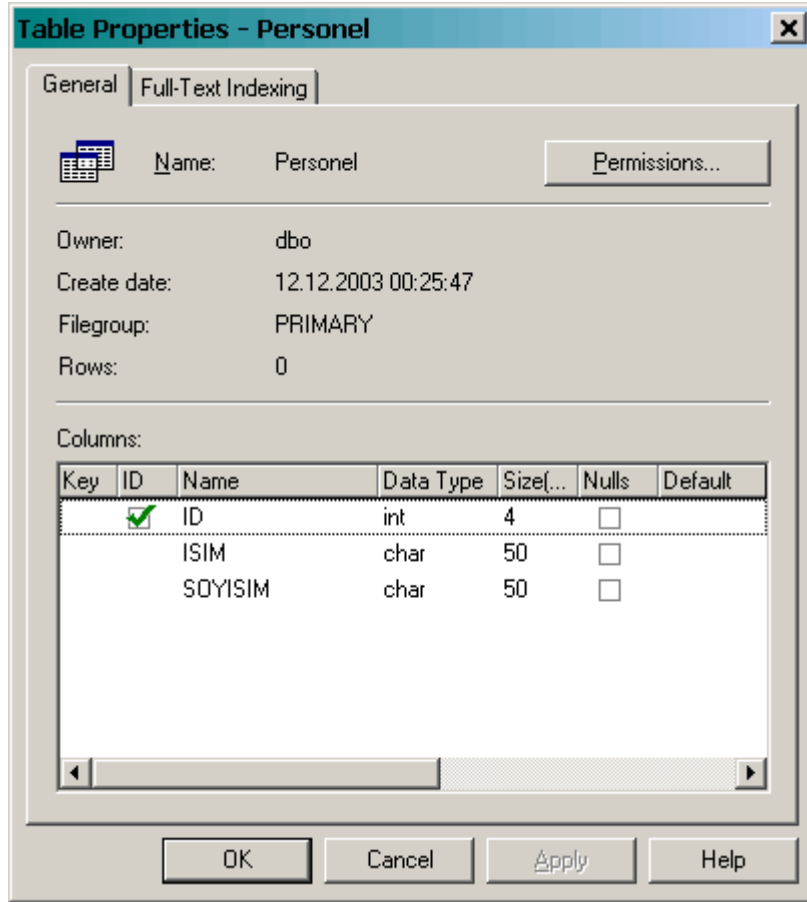
Şekil 4. Veritabanımız Oluşturuldu.

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



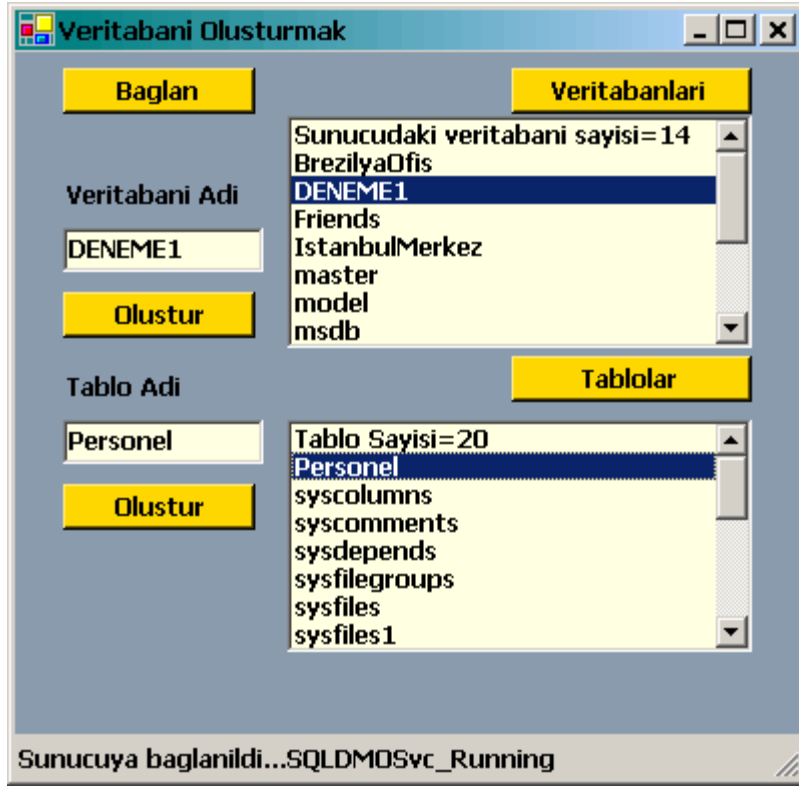
Şekil 5. Veritabanımız ve Tablomuz oluşturuldu.

Ve tablomuza baktığımızda oluşturduğumuz alanlarda 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 tablolarıda gelir. Bununla birlikte veritabanları görünürken sadece TempDBd 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 - 15 Aralık 2003 Pazartesi

ado.net, dataview,

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 içersinde 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 görünümleri 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 kontollere 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. İşlemler.

Şimdi gelin konuyu daha iyi anlayabilmek amacıyla örneklerimize geçelim. Konuyu anlayabilmek için iki farklı örnek yapacağız. İlk örneğimizde, aynı DataTable için farklı görünüm 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");
```

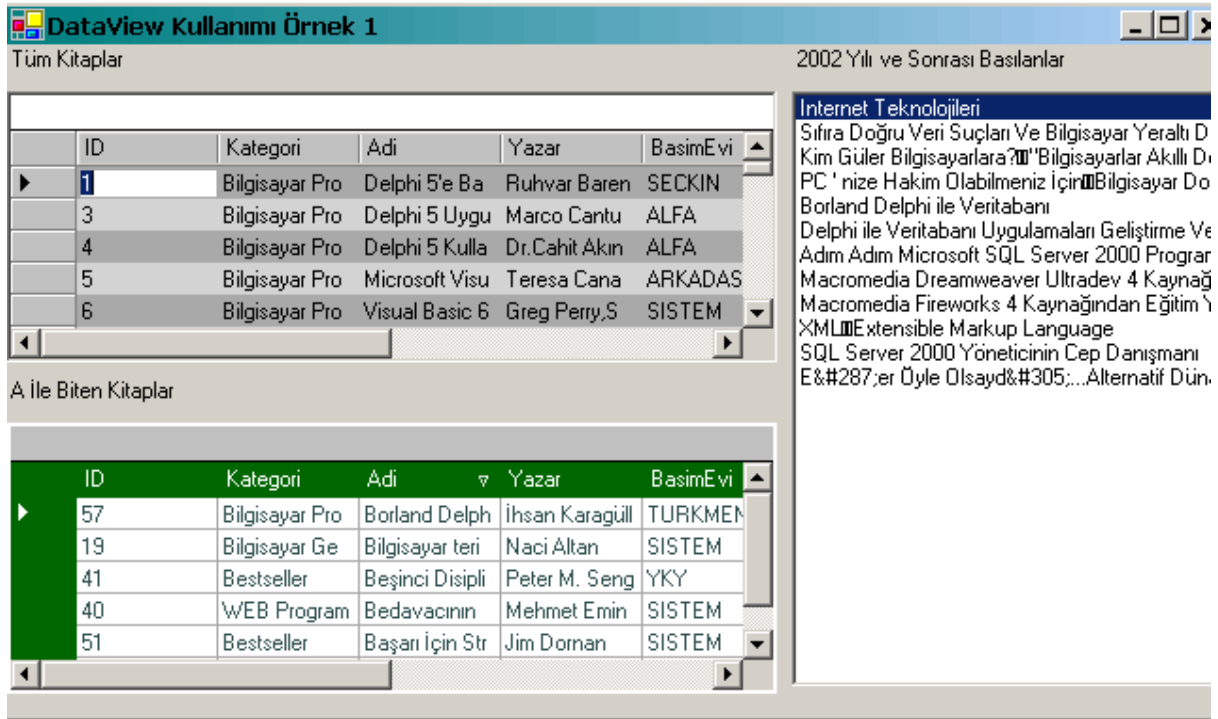
```
    da.Fill(dtKitaplar);
```

```

}
private void Form1_Load(object sender, System.EventArgs e)
{
    Baglan();
    DataView dvTum_Kitaplar=dtKitaplar.DefaultView; /* Bir DataTable nesnesi yaratıldığı
    zaman, standart olarak en az bir tane görünüme sahiptir. Bu varsayılan görünüm bir
    DataView nesnesi döndüren DefaultView metodu ile elde edilebilir. Çalıştırdığımız sql
    sorgusu Kitaplar tablosundaki tüm kayıtları aldığından buradaki DefaultView'da aynı veri
    kümesini sahip olacak bir DataView nesnesi döndürür. Biz bu dönen veri kümesini
    dvTum_Kitaplar isimli DataView nesnesine aktardık. Daha sonra ise DataView nesnemizi
    dgTum_Kitaplar isimli dataGrid nesnemize bağladık.*/
    dgTum_Kitaplar.DataSource=dvTum_Kitaplar;
    /* Yeni bir DataView 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, DataViewRowState türünden bir parametre. Bu özellik DataView
    içerisinde ye alan her bir DataRowView'un (yani satırın) durumunun değerini belirtir. Alacağı
    değerler
    * 1. Added ( Sadece DataView'a eklenen satırları ifade eder)
    * 2. Deleted ( Sadece DataView'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 orijinal 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 DataView 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";
}

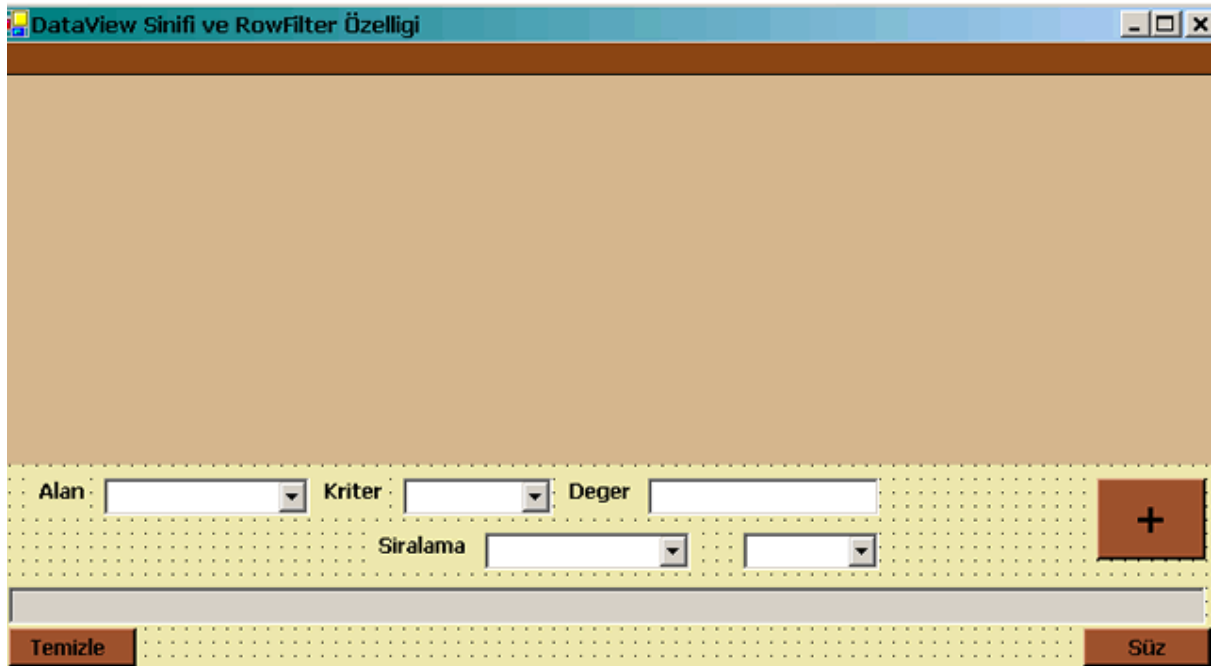
```

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 nesnemizin bellekte temsil
ettiği tablonun Alanlari ile doldurur. Nitekim bu alanlari, 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 ettiği 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, parametre olarak DataTable nesnemizi alıyor. Dolayısıyla DataView
nesnemiz, dataTable içindeki veriler ile dolmuş şekilde oluşturuluyor.*/
    dataGrid1.DataSource=dvKitaplar; /* DataGridView kontrolümüze veri kaynağı olarak,
DataView nesnemizi işaret ederek, DataView içindeki verileri göstermesini sağlıyoruz.*/
    AlanDoldur();
}
/* Bu butona bastığımızda, kullanıcının seçtiği alan, filtreleme kriteri ve filtreleme için
kullanılacak değer 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 koşullu ifadelerinde, seçilen alanın veri tipine bakıyoruz. Nitekim RowFilter
metodunda, alan'ın veri tipine göre ifademiz değişiklik gösteriyor. Tarih tipindeki verilerde #
karakteri aranan metnin başına ve sonuna gelirken, string tipinde değerlerde ' karakteri
geliyor. Sayısal tipteki değerler için ise herhangi bir karakter ifadenin aranan değer başına
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; /* Olusturulan 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 gerçekleştirmis oluyoruz. */
    dvKitaplar.Sort=cmbAlanSira.Text+" "+cmbSiralamaKriteri.Text; /* Burada ise Sort
özelligine 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' sınırın 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 - 16 Aralık 2003 Salı

ado.net, datagrid, paging,

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 alacak kolonları
kendimiz ayarlayacağımız için bu özelliğe false değerini aktardık.*/
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 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 çalıştırırsak aşağıdakine benzer bir ekran görüntüsü ile karşılaşırız.

Kitap Listesi

No	Kategorisi	Kitap	Yazari
47	Bestseller	2500 Yıllık Savaş Tarihi	Jhon Keegar
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
        olacak. Yani 1 den başlayıp kaç sayıtlık sayfa oluştuysa o kadar sayıda bir buton dizesi
        dataGrid kontrolünün en altında yer alıcak.*/

        dgKitap.AutoGenerateColumns=false; /* DataGrid kontrolümüzde yer alıcak kolonları
        kendimiz ayarlayacağımız için bu özelliğe 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.

1 2 3 4 5 6

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="FFFFFF" BackColor="#990000"></HeaderStyle>
    <FooterStyle ForeColor="#330099" BackColor="FFFFFF"></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="FFFFFF"></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 - 17 Aralık 2003 Çarşamba

ado.net, datagrid,

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="Siral" 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 Fiyatı"></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 nesnemizin 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 yapabilmesini sağlamış oluyoruz. */
```

```
}
```

```
/* Sirala isimli metodumuz, DataGrid tagında OnSortCommand için belirttiğimiz metoddur.
```

Bu metod ile , bir kolon başlığına tıkladığında 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 nesnemizin 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 Siralama(object sender,DataGridSortCommandEventArgs e)
{
    lblSiralamaKriteri.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 : Adi

ID	Kıtap	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 E
13	Adım Adım Web Veritabanı Geliştirme	Jim Buyea
64	Adım Adım XML	Michael J
12	Analiz, tasarım ve Uygulamalı Sistem Yönetimi	Prof.Dr.E
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	Bedavacının Web Sitesi Tasarım Kılavuzu Bir Kuruş Bile Harcamadan Web Sitesi Sahibi Olun	Mehmet I

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. Cah
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 na 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 - 18 Aralık 2003 Perşembe

C#,

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 tutulmaktadır. Tüm koleksiyon sınıflarının ortak özelliği barındırdıkları verileri object tipinde olmalarıdır. 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 tasarımını 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 lstAnahtar_DoubleClick(object sender, System.EventArgs e)
```

```
{
```

```
    string deger;
```

```
deger=htTeknikSozluk[lstAnahtar.SelectedItem.ToString()].ToString();
```

```
/* Hashtable'daki bir değere ulaşmak için, köşeli parantezler arasında aranacak key değerini giriyoruz. Sonucu bir string değişkenine aktarıyoruz.*/
```

```
MessageBox.Show(deger,lstAnahtar.SelectedItem.ToString());
```

```
}
```

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

```
{
```

```
    if(htTeknikSozluk.Count==0)
```

```
    {
```

```
        stbDurum.Text="Çıkarılabilecek hiç bir eleman yok";
```

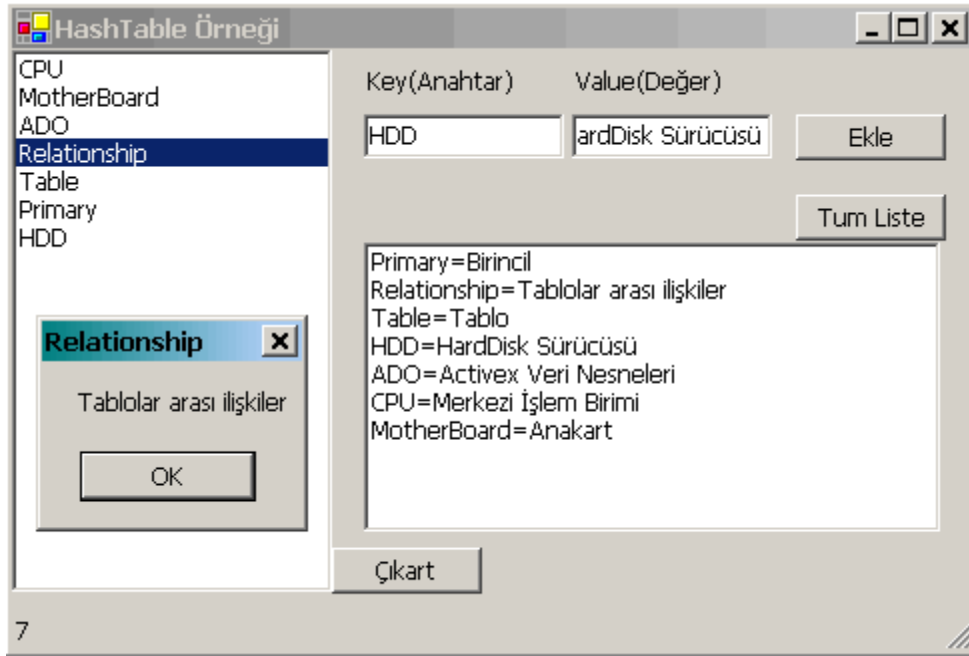
```

    }
    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="Çıkartıldı";
        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.

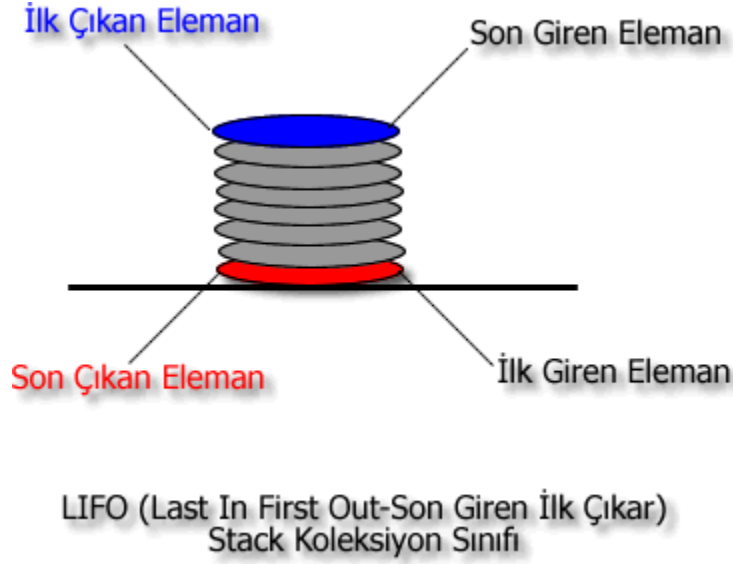
Stack ve Queue Koleksiyon Sınıfı - 19

Aralık 2003 Cuma

C#,

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 prensipleri 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.



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;



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 önüce geçen metod Peek metodudur. Şimdi diyebilirsinizki 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 elemene 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 ediyoruzç. 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.*/
        }
    }
}
}

```

```

C:\ "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
süreçe 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.*/
}
}
}
}

```

```

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 - 22 Aralık 2003 Pazartesi

C#,

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;

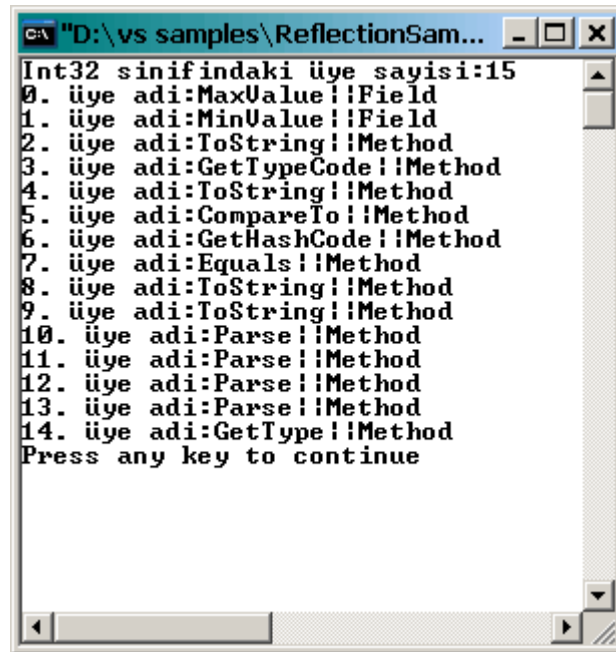
```
using System;
namespace ReflectionSample1
{
    class Class1
    {
        static void Main(string[] args)
        {
            Type tipimiz = Type.GetType("System.Int32");
            /* Öncelikle String sınıfının tipini öğreniyoruz. */
            System.Reflection.MemberInfo[] tipUyeleri = tipimiz.GetMembers();
            /* 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 dizininde 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;

ConstructorIn

Tipe ait yapıcı metod üyelerini ve bu üyelere ait

fo	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 erişebilecez.*/
            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;

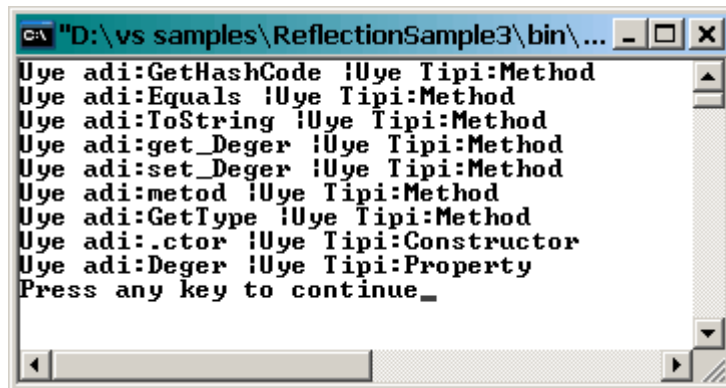
```

```

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 yazdığımız sınıf üyelerinede bakabiliriz.

Peki kendi sınıfımıza ait bilgileri edinmenin bize ne gibi bir faydası olabilir. İşte şimdi tam dişimize 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, özelliğimizin 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 saklanmasını 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.

Adi	BasimEvi
Delphi 5'e Bakış	SECKIN
Delphi 5 Uygulama Geliştirme Kılavuzu	ALFA
Delphi 5 Kullanım Kılavuzu	ALFA
Microsoft Visual Basic 6.0 Geliştirmek Ust	ARKADAS
Visual Basic 6 Temel Başlangıç Kılavuzu	SISTEM
Microsoft Visual Basic 6 Temel Kullanım K	ALFA
ASP ile E-Ticaret Programcılığı	SISTEM
ASP 3.0	SISTEM
ASP ile web programcılığı ve elektronik tic	PUSULA
Herkes için ASP ile Veritabanı Yönetimi 3.	ALFA

Doldur

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 iseler yazılabilir olup olmadıklarını öğrenmekte kullandık. Değerli Okurlarım. Geldik bir makalemizin daha sonuna. Hepinize mutlu günler dilerim.

Bir Sınıf Yazalım - 23 Aralık 2003 Salı

C#,

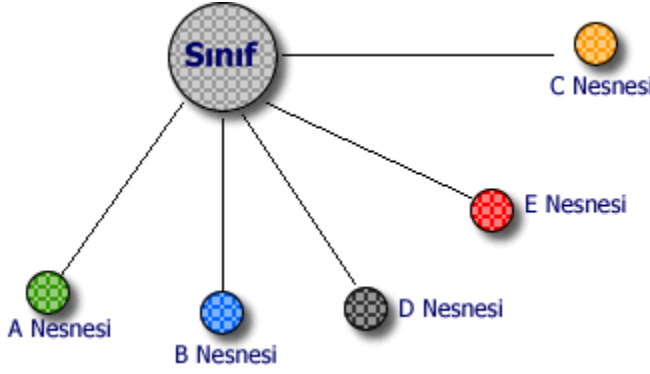
Değerli Okurlarım, Merhabalar.

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 bileneğiniz 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.

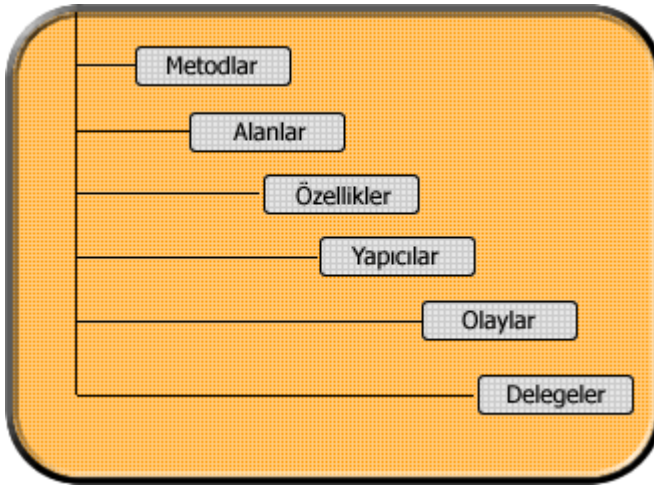
Her çiçeğin bir renginin(renklerinin) olması gibi. İşte nesne yönelimli programlama kavramında bahsedilen ve her şeyin temelini oluşturan sınıf kavramı bu benzetme ile tamamen aynıdır. Çiçek bir sınıf olarak algılanırken, sokakta gördüğümüz her çiçek bu sınıfın ortak özelliklerine sahip birer nesne olarak 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 ilerleyen 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 örneklenirilen nesneler için farklı değerlere sahip olurlar. Yani bir sınıf varken, bu sınıftan örneklenirilmiş 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 yapıcıları kullanırız. Özellikler kapsülleme dediğimiz Encapsulating kavramının bir parçasıdır. Çoğunlukla, sınıf içersinden tanımladığımız alanlara, dışardan 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 derseniz 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 özelliğin 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 içersinde 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 göndermiyecek 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
içerisinden çağırılabiliriz 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 akatarıyoruz.*/
    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,txtVeritabani.Text,txtKullanici.Text,txtParola.Text);
```

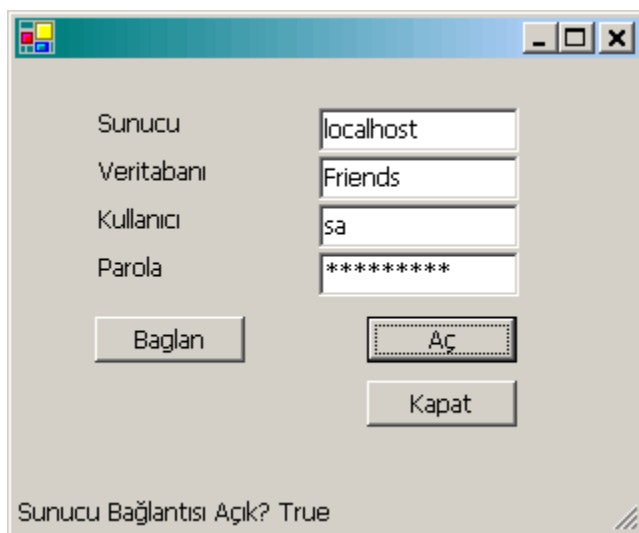
```
Veriler.Verı v=new Veri(  
    ▲ 2 of 2 ▼ Veri.Verı (string sunucuAdi, string veritabaniAdi, string kullaniciAdi, string sifre)
```

```
})
```

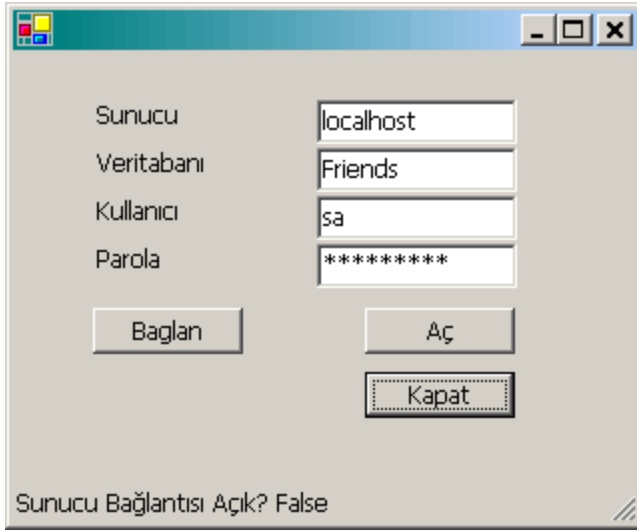
```
private void btnAc_Click(object sender, System.EventArgs e)  
{  
  
    v.baglantiDurumu= true;  
    stbDurum.Text="Sunucu Bağlantısı Açık? "+v.baglantiDurumu.ToString();  
  
}
```

```
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) Metotlar - 25 Aralık 2003 Perşembe

C#,

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()
    {

    }

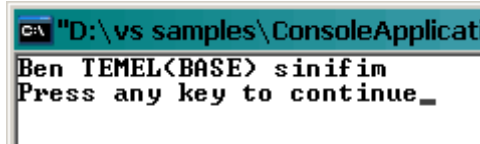
    public void Yazdir()
    {
        Console.WriteLine("Ben TUREYEN(DERIVED) sinifim");
    }
}

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 çağırılmasındadır. Biz burada Tureyen sınıf nesnesini aktardığımız için bu sınıfa ait Yazdir metodunun çalıştırılmasını bekleriz. Oysaki sonuç aşağıdaki gibi olacaktır.



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()
        {
        }

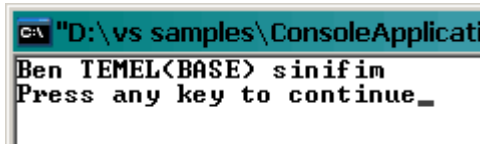
        public new void Yazdir()
        {
            Console.WriteLine("Ben TUREYEN(DERIVED) sinifim");
        }
    }

    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 Yazdir 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();
}
}
}

```



Şekil 3. Tureyen sınıftaki Yazdir metodu çağırılmıştır.

Burada önemli olan nokta, Temel sınıfaki metodun virtual anahtar kelimesi ile tanımlanması, Tureyen sınıftaki 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 - 25 Aralık 2003 Perşembe

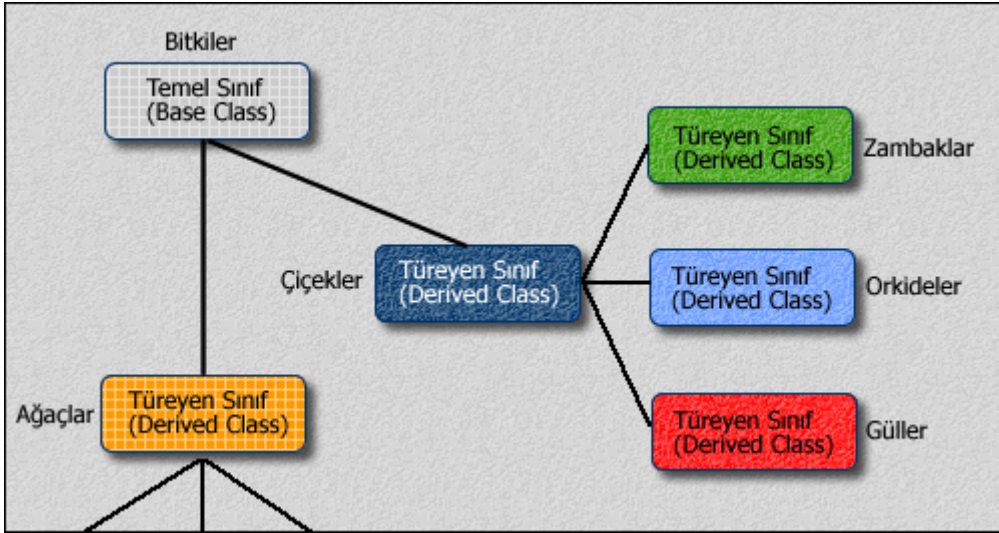
C#,

Değerli Okurlarım, Merhabalar.

Bir önceki makalemizde C# dilinde sınıf kavramına bir giriş yapmış ve OOP(Object 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 türetilen 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 kalıtımda önemli role sahip base ve new anahtar kelimelerinin kullanımında göreceksiniz. Base ile temel sınıfa nasıl parametere 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 özelliklerindeki 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 aynıysa 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 TemelSinif'a 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 Seklim  
dikdortgen  
Benim Seklim Alanı:8  
Benim Seklim 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 - 28 Aralık 2003 Pazar

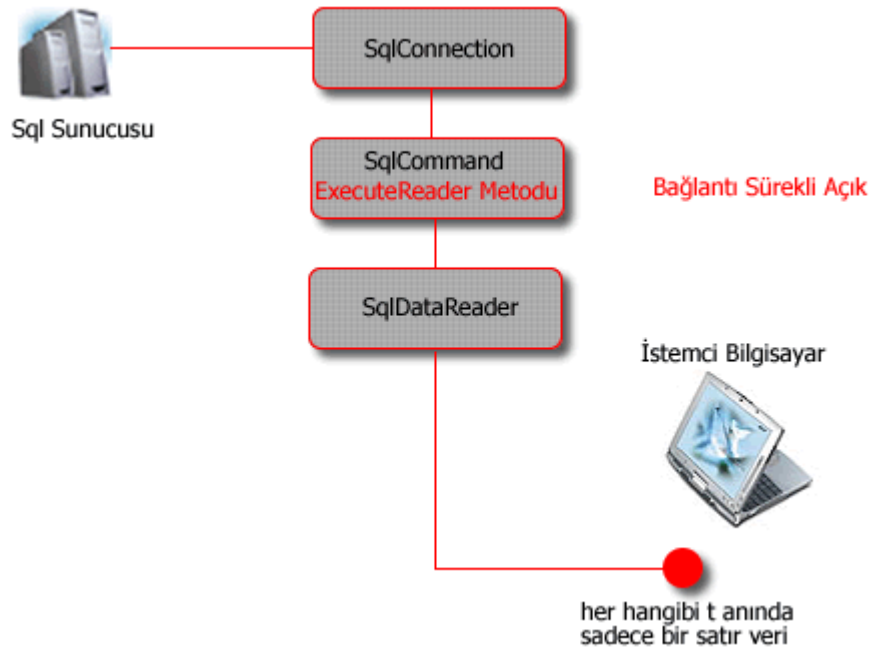
ado.net, sqldatareader,

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 nahoş 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 tasarlanmış. 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.

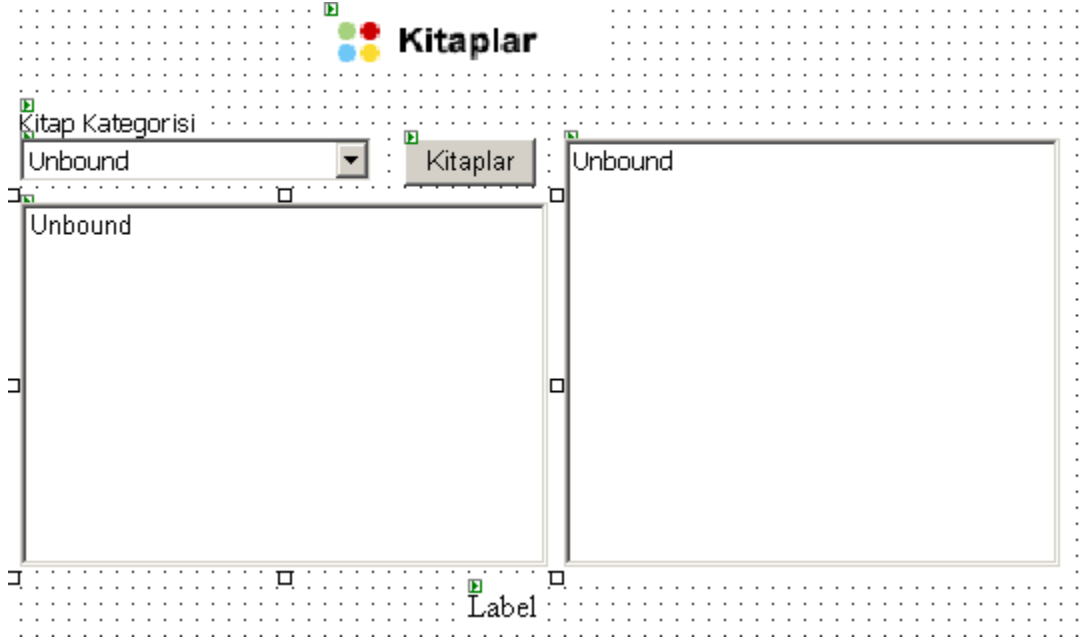
Görüldüğü gibi sql sunucusuna sürekli bağlı durumdayız. Bunun yanında SqlCommand sorgusunun çalıştırılması sonucu elde edilen veri akımı SqlDataReader nesnesince temsil edilmektedir. İstemci bilgisayarımız, her hangibir t zamanında kendi belleğinde SqlDataReader'ın taşıdığı veri akımının sadece tek bir 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 bir 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 sağlar. 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 yapılabilmesinde 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, 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 çalıştıracığımız 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. */

```
}
```

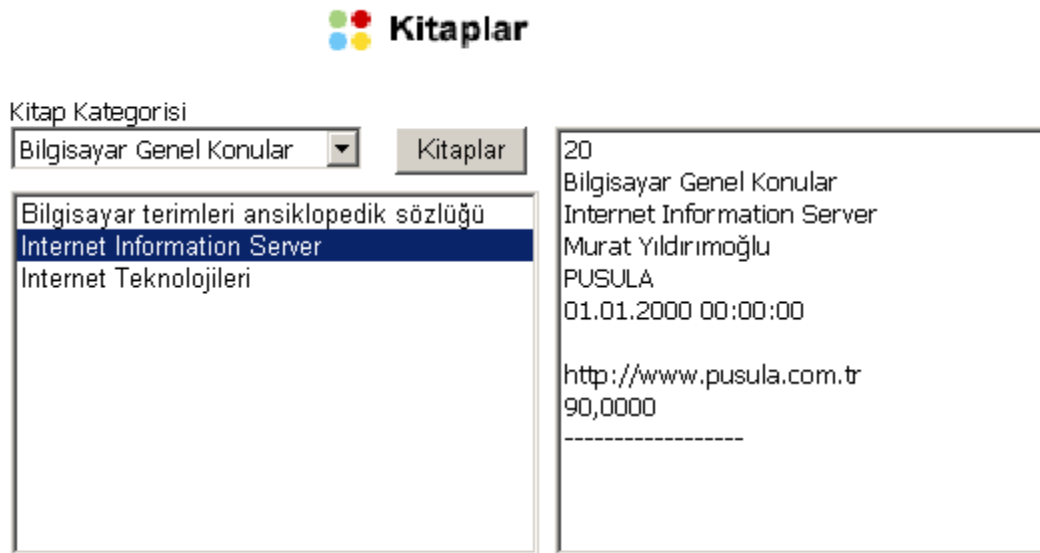
```
private void IstKitaplar_SelectedIndexChanged(object sender, System.EventArgs e)
```

```

{
    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 - 29 Aralık 2003

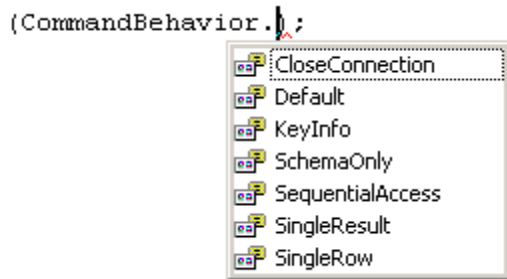
Pazartesi

ado.net, sqldatareader,

Değerli Okurlarım, Merhabalar.

Bir önceki makalemizde SqlDataReader sınıfını incelemeye başlamıştık. Listeleme amaçlı veri kümelerinin görüntülemesinde 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.



Şekil 1. CommandBehavior Davranışları

CommandBehavior Değeri	İşlevi
------------------------	--------

CommandBehavior.CloseConnection	SqlDataReader nesnesi Close metodu ile kapatıldığında, ilişkili SqlConnection nesneside otomatik olarak kapatılır. Nitekim, işlemiz 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 davranışı 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 tanıırken performansıda arttırmaktadır.
CommandBehavior.KeyInfo	Bu durumda sql sorgusu sonucunda SqlDataReader nesnesi, tabloya ait anahtar alan bilgisini içerir.

Tablo 1. CommandBehavior Davranışları

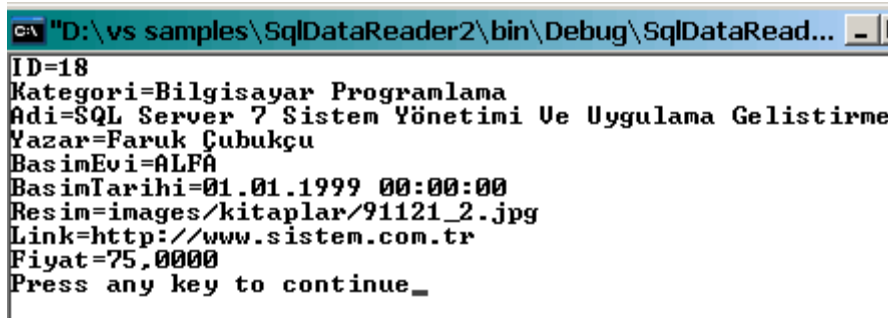
Şimdi dilerseniz basit Console uygulamaları ile, yukarıdaki davranışların işleyişlerini inceleyelim. CommandBehavior. CloseConnection durumunu önceki

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();  
        }  
    }  
}
```



```
C:\>"D:\vs samples\SqlDataReader2\bin\Debug\SqlDataRead...
ID=18
Kategori=Bilgisayar Programlama
Adi=SQL Server 7 Sistem Yönetimi Ve Uygulama Gelistirme
Yazar=Faruk Çubukçu
BasimEvi=ALFA
BasimTarihi=01.01.1999 00:00:00
Resim=images/kitaplar/91121_2.jpg
Link=http://www.sistem.com.tr
Fiyat=75,0000
Press any key to continue_
```

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();
        }
    }
}
```



```
C:\>"D:\vs samples\SqlDataReader3\bi
Ortalama Birim Fiyati=28,8663
Press any key to continue_
```

Şekil 3. SingleResult davranışı.

Şimdiye 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.

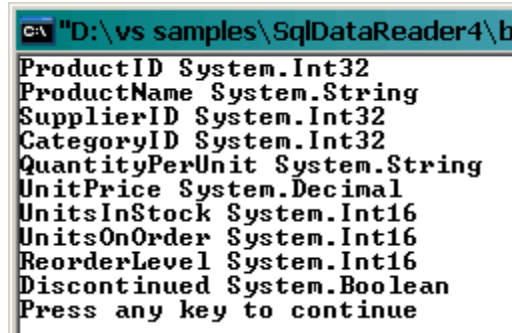


Şekil 4. Hata.

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. İkinci parametre bu alanın kaçınıcı karakterinden itibaren okunmaya başlanacağı bildirir. Üçüncü parametre ise `char` tipinden bir diziyi belirtir. Okunan her bir karakter bu diziye aktarılacaktır. Dördüncü parametre ile dizinin kaçınıcı elemanından itibaren aktarımın yapılacağı belirtilir. Son parametremiz ise ne kadar karakter okunacağını belirtmektedir. Şimdi kodlarımızı yazalım.

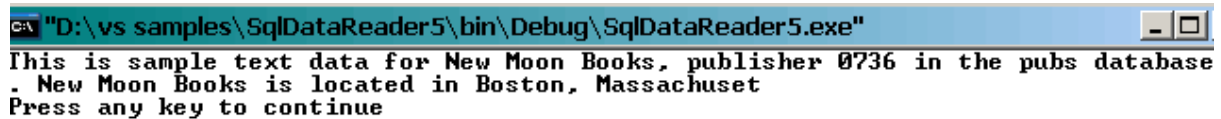
```
using System;
using System.Data;
using System.Data.SqlClient;
namespace SqlDataReader5
{
    class Class1
    {
        static void Main(string[] args)
        {
            SqlConnection conPubs=new SqlConnection("data source=localhost;integrated
            security=sspi;initial catalog=pubs");
```

```

        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) - 30 Aralık 2003 Salı

Ado.Net,

Değerli Okurlarım, Merhabalar.

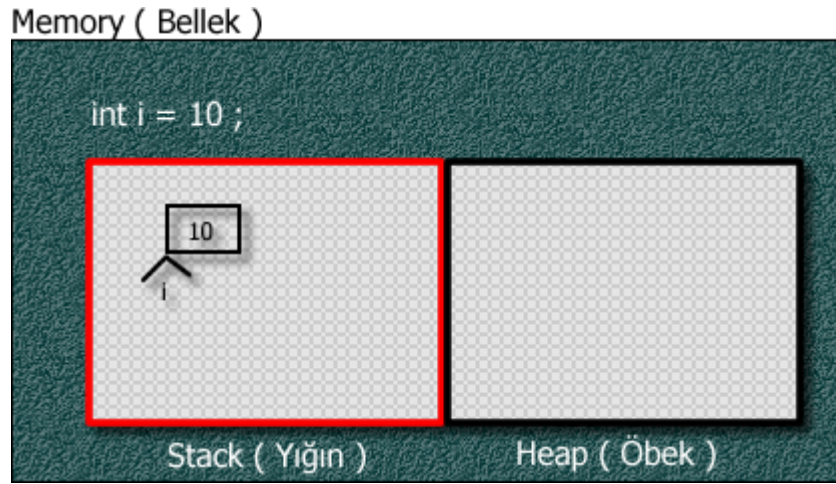
Bugünkü makalemizde, Boxing ve Unboxing kavramlarını incelemeye çalışacağız. Boxing değer türü bir değişkeni, referans türü bir nesneye aktarmaktır. Unboxing işlemi ise bunun tam tersidir. Yani referans türü değişkenin işaret ettiği değeri tekrar , değer türü bir değişkene aktarmaktır. Bu tanımlarda 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 bölegesinde nasıl tutulduğuna ilişkin açıklayıcı şekli de aşağıda görebilirsiniz.

Value type (Değer Tipler)	
bool	long
byte	sbyte
char	short
decimal	Struct (Yapılar)
double	uint
Enum (Numaralandırıcılar)	ulong
float	ushort
int	

Tablo 1. Değer Tipleri



Şekil 1. Değer Tiplerinin Bellekte Tutuluşu

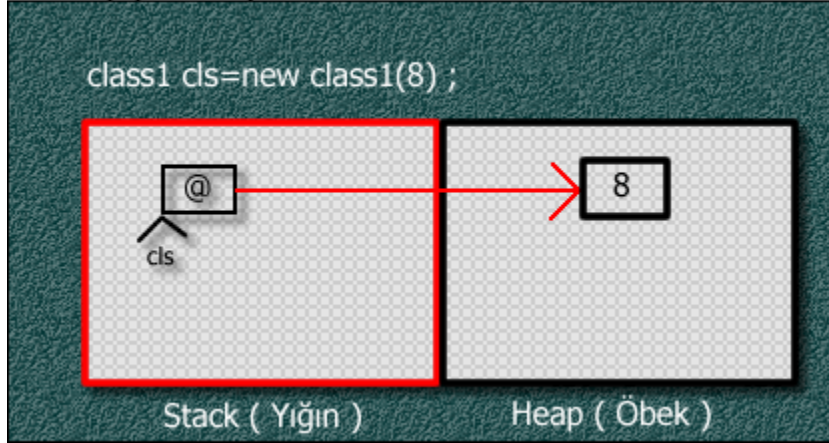
Şekildende görüldüğü gibi, Değer Türleri bellekte, Stack dediğimiz bölgede tutulurlar. Şimdi buraya kadar anlaşılmayan bir şey yok. İlginç olan reference(başvuru) tiplerinin bellekte nasıl tutulduğudur. Adındanda anlaşıldığı gibi reference tipleri asıl veriye bir başvuru içeriler. Örneğin sınıflardan 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 değer 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;

```

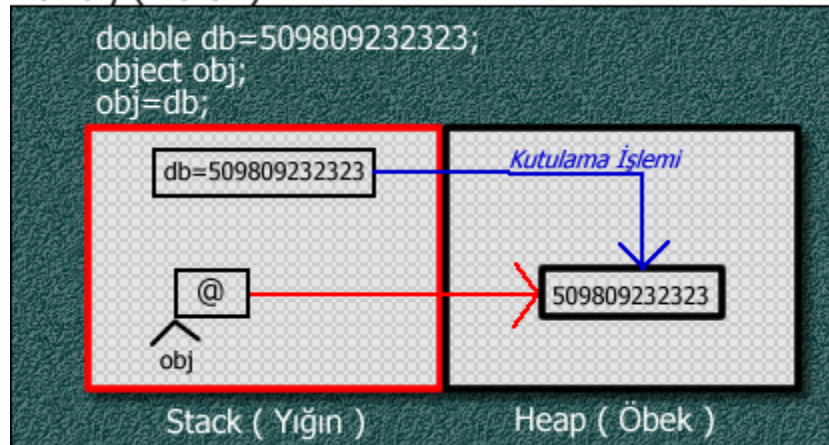
C:\ "D:\vs samples\boxunbox\bir
509809232323
509809232323
509809232324
509809232323
Press any key to continue_

```

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 artırım db değişkeninin yığında yer alan orjinal değeri üzerinde gerçekleşmektedir. Bu işlemi açıklayan şekil aşağıda yer almaktadır.

Memory (Bellek)



Şekil 4. Boxing İşlemi

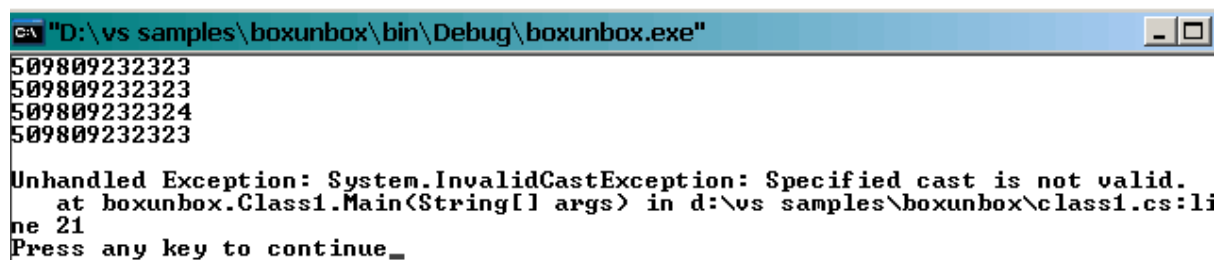
Boxing işlemi otomatik olarak yapılan bir işlemdir. Ancak UnBoxing işleminde durum biraz daha değişir. Bu kez , başvuru nesnemizin işaret ettiği veriyi öbekten alıp yığındaki bir değer tipi alanı olarak kopyalanması söz konusudur. İşte burada tip uyumsuzluğu denen bir kavramla karşılaşırız. Öbekten , yığına

kopylanacak olan verinin, yığılma 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 değeri 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 referans tipimizin öbekte başvurduğu veri tipi integer bir değeri 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
```

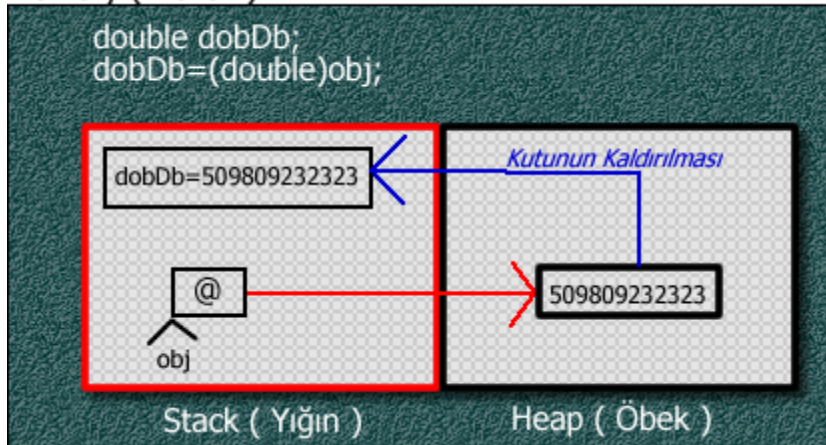
```

{
    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.