C Sharp Programlama Dili/Veri tabanı işlemleri

Vikikitap, özgür kütüphane

C#'ta veri tabanı işlemleri System.Data isim alanındaki ve bu isim alanının altındaki alt isim alanlarındaki türlerle yapılır. System.Data isim alanına programcılar ADO.NET ismini vermişlerdir. Yani bir yerde ADO.NET duyarsanız aslında System.Data isim alanından bahsetmektedir. System.Data isim alanıyla veri tabanlarına iki şekilde erişilebilir.

1. MSSQL veri tabanlarına direkt erişim.
2. OLEDB protokolünü destekleyen veri tabanlarına OLEDB protokolü ile erişim.

Tabii ki bağlanacağınız veri tabanı MSSQL ise daha hızlı olması açısından birinci yöntemi seçmeniz tavsiye edilir. Ancak daha genel bir yol olduğundan bu bölümde ikinci yöntem üzerinde daha çok duracağız. Popüler tüm veri tabanları OLEDB protokolünü desteklemektedir. OLEDB protokolüyle MSSQL veri tabanlarına da erişebilirsiniz. Ayrıca OLEDB protokolüyle Access dosyalarına da bir veri tabanıymış gibi bağlanabilirsiniz.

Şimdi isterseniz System.Data isim alanı ve bu isim alanındaki alt isim alanları hakkında kısa bilgiler verelim:
System.Data Veri tabanlarındaki verilerle çalışmak için gerekli temel türler bu isim alanındadır. Veri tabanlarına bağlanmak için gerekli türler bu isim alanında değildir.
System.Data.Common İleride göreceğiz.
System.Data.OleDb OLEDB protokolünü destekleyen veri tabanlarına bağlanmak için gerekli türler barındırır.
System.Data.SqlClient OLEDB kullanmadan direkt MSSQL veri tabanlarına bağlanmak için gerekli türler barındırır.
System.Data.SqlTypes MSSQL veri tabanlarındaki veri türlerini içerir. Tabii ki veri tabanından veri çekerken veya veri tabanına veri kaydederken C#'a özgü veri türlerini (string, int, ...) kullanabiliriz. Ancak MSSQL'e özgü veri türlerini kullanmamız bize artı performans sağlar.

Veri tabanına bağlanma

Burada çeşitli veri tabanlarına bağlanma hakkında üç örnek verilecektir:

```
//veri kaynağına erişmek için çeşitli bilgiler hazırlanır.
string kaynak="Provider=SqlOleDb;server=SunucuAdı;uid=KullanıcıAdı;pwd=Şifre
//kaynak stringi kullanılarak bağlantı nesnesi oluşturulur.
OleDbConnection baglanti=new OleDbConnection(kaynak);
//OleDbConnection sınıfının static olmayan Open() metoduyla bağlantı aktif h
baglanti.Open();
```

Bu örneğimizde OLEDB protokolünü destekleyen bir veri tabanına bağlandık. OleDbConnection sınıfı System.Data.OleDb isim alanındadır.

```
string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb";
OleDbConnection baglanti=new OleDbConnection(kaynak);
baglanti.Open();
```

Burada ise OLEDB protokolünü kullanarak bir Access dosyasına bir veri tabanıymış gibi bağlandık. Gördüğünüz gibi öncekinden tek farkı kaynak stringi.

```
string kaynak="server=SunucuAdı;uid=KullanıcıAdı;pwd=Şifre;database=VeriTaba
SqlConnection baglanti=new SqlConnection(kaynak);
baglanti.Open();
```

Burada ise OLEDB protokolünü kullanmadan direkt olarak bir MSSQL veri tabanına bağlandık. SqlConnection sınıfı System.Data.SqlClient isim alanındadır. Biz ders boyunca ikinci yöntemi kullanacağız. Yani Access dosyalarına erişeceğiz. OLEDB protokolünü kullanarak normal bir veri tabanına bağlanıp işlem yapma ile aynı şeyi Access dosyalarına yapma arasındaki tek fark bağlanma şeklidir. Yani veri tabanına veya Access dosyasına OLEDB protokolü ile bir kere bağlandıktan sonra veri tabanından veri çekme/veri tabanına veri kaydetme vb. işlemler tamamen aynıdır. OLEDB protokolünü kullanmadan bağlanılan MSSQL veri tabanlarıyla işlemler yapmak kısmen farklı olsa da işlemler büyük ölçüde aynı mantıkla yapılır. Şimdi isterseniz bir veri tabanına bağlanma örneği yapalım:

```
using System;
using System.Data.OleDb;
class vt
{
    static void Main()
    {
        string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb"
        OleDbConnection baglanti=new OleDbConnection(kaynak);
        baglanti.Open();
        Console.WriteLine("Bağlantı sağlandı...");
        baglanti.Close();
    }
}
```

Bu örneğin hata vermeden çalışabilmesi için programımızla aynı klasörde dosya.mdb dosyasının olması gerekir. Access'te bu dosyayı oluşturun. Dosyamız çeşitli alanları olan bir tablo içersin, bu alanlara da veri türleriyle uyumlu olmak şartıyla istediğiniz verileri koyun. OleDbConnection sınıfının static olmayan Close() metodu ilgili bağlantıyı kesip bağlantının kullandığı sistem kaynaklarının serbest bırakılmasını sağlar. Eğer bağlantı sağlanmışsa programın ekrana `Bağlantı sağlandı...` çıktısını vermesi gerekir. Eğer bağlantı sağlanamamışsa istisnai durum oluşması gerekir.
ÖNEMLİ NOT: Bu ders boyunca sizin SQL dilini ve Access programını başlangıç seviyesinde bildiğiniz varsayılmıştır. Örneklerde olabildiğince basit SQL cümleleri kullanılmıştır.
NOT: System.Data isim alanı şimdiye kadar gördüğümüz isim alanlarının aksine mscorlib.dll assemblysinde değil, System.Data.dll assemblysindedir. Bazı .Net Framework sürümlerinde derleyici bu dosyayı otomatik olarak her dosyaya refere etmesine rağmen bazı .Net Framework sürümlerinde refere etmemektedir. Eğer program derleme hatası verirse bir de bu dosyayı refere etmeyi deneyin.

OleDbCommand sınıfı

OleDbCommand sınıfı bir veri tabanına komut göndermek için kullanılır. OleDbCommand sınıfı System.Data isim alanındadır. Bir veri tabanına gönderilmesi için üç şekilde komut oluşturabiliriz.

```
string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb";
OleDbConnection baglanti=new OleDbConnection(kaynak);
baglanti.Open();
string sorgu="select * from tablo";
OleDbCommand komut=new OleDbCommand(sorgu,baglanti);
```

Bu örneğimizde OleDbCommand sınıfının yapıcı metodu bir OleDbConnection ve bir de string nesnesi aldı. String nesnesi SQL cümlesini içeriyor. Şimdi ikinci yönteme geçelim.

```
string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb";
OleDbConnection baglanti=new OleDbConnection(kaynak);
baglanti.Open();
OleDbCommand komut=new OleDbCommand("Tablo",baglanti);
komut.CommandType=CommandType.TableDirect;
```

CommandType System.Data isim alanında bulunan bir enumdur. OleDbCommand sınıfının static olmayan CommandType özelliğinin tipi CommandType enumudur. Bu örneğimizde veri tabanına bir SQL cümlesi göndermektense bir tabloyu tamamen programa çekmek istedik.

```
string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb";
OleDbConnection baglanti=new OleDbConnection(kaynak);
baglanti.Open();
OleDbCommand komut=new OleDbCommand("StorProsedur",baglanti);
komut.CommandType=CommandType.StoredProcedure;
komut.Parameters.Add("@Yas","30");
```

Bu örnekte ise bir stor prosedür çalıştırdık. Parametreyi de Parameters alt sınıfındaki Add() metoduyla verdik.
NOT: Bu üç örneğimizde OleDbConnection nesnesini ve komutu yapıcı metotta verdik. İstersek şöyle de verebilirdik:

```
string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb";
OleDbConnection baglanti=new OleDbConnection(kaynak);
baglanti.Open();
string sorgu="select * from tablo";
OleDbCommand komut=new OleDbCommand();
komut.Connection=baglanti;
komut.CommandText=sorgu;
```

NOT: OleDbCommand sınıfının CommandType özelliğine CommandType.Text ilk değeri verilmiştir. Bu da komutun bir SQL cümlesi olduğunu belirtir.
Şimdiye kadar güzel. Ancak halen bazı şeyler eksik. Artık komut elimizde var. Ancak halen komutu çalıştırmadık. Belli bir komutu (OleDbCommand nesnesini) çalıştırmak için OleDbCommand sınıfının 4 farklı static olmayan metodu vardır.
int ExecuteNonQuery() Bu metot veri tabanındaki kayıtlarda değişiklik (ekleme-silme-değiştirme) yapan komutları çalıştırmak için kullanılır. Metot tabloda etkilenen (silinen-eklenen-değiştirilen) kayıt sayısını döndürür. Örnek:

```
string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb";
OleDbConnection baglanti=new OleDbConnection(kaynak);
baglanti.Open();
string sorgu="INSERT INTO Tablo (Ad,Soyad) VALUES ('Mehmet','Kaplan')";
OleDbCommand komut=new OleDbCommand(sorgu,baglanti);
Console.WriteLine(komut.ExecuteNonQuery()+" tane ekleme yapıldı.");
baglanti.Close();
```

object ExecuteScalar() Bu metot veri tabanından tek bir veri elde eden komutları çalıştırmak için kullanılır. Örneğin SQL dilinin COUNT deyimi tablodaki kayıt sayısını verir. Örneğin `SELECT COUNT(*) FROM Tablo` SQL cümlesini çalıştırmak için bu metot kullanılabilir. Örnek:

```
string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb";
OleDbConnection baglanti=new OleDbConnection(kaynak);
baglanti.Open();
string sorgu="SELECT COUNT(*) FROM Tablo";
OleDbCommand komut=new OleDbCommand(sorgu,baglanti);
Console.WriteLine((int)komut.ExecuteScalar()+" tane kayıt var.");
baglanti.Close();
```

OleDbDataReader ExecuteReader() Bu metot veri tabanından bir tablo çeken komutları çalıştırmak için kullanılır. Genellikle SELECT deyimleri bu metotla çalıştırılır. Bir OleDbDataReader nesnesi döndürür.

OleDbDataReader sınıfı

OleDbCommand sınıfının static olmayan ExecuteReader() metodu ile oluşan tabloları tutmaya yarayan bir sınıftır. OleDbDataReader nesneleri new operatörüyle oluşturulamaz. OleDbDataReader nesneleri oluşturmak için OleDbCommand sınıfının static olmayan ExecuteReader() metodunu kullanabiliriz. OleDbDataReader sınıfı System.Data.OleDb isim alanındadır. Şimdi isterseniz Access'te şöyle bir tablo oluşturalım:

id	ad	soyad	not
1	ali	yılmaz	70
2	mehmet	süzen	90

3	zafer	kaplan	100
4	mehmet	oflaz	45
5	ayşe	yılmaz	34
6	hatice	özdoğan	100
7	emine	şanlı	20

Bu tabloya ogrenci ismini verelim. Dosyayı kaynak kodumuzla aynı klasörde oluşturup adını dosya.mdb koyalım. Şimdi aşağıdaki programı yazalım.

```
using System;
using System.Data;
using System.Data.OleDb;
class vt
{
    static void Main()
    {
        string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb"
        OleDbConnection baglanti=new OleDbConnection(kaynak);
        baglanti.Open();
        string sorgu="SELECT * FROM ogrenci";
        OleDbCommand komut=new OleDbCommand(sorgu,baglanti);
        OleDbDataReader cikti=komut.ExecuteReader();
        string format="{0,-10}{1,-10}{2,-10}";
        Console.WriteLine(format,"Adı","Soyadı","Notu");
        Console.WriteLine("".PadRight(30,'-'));
        while(cikti.Read())
        {
            Console.WriteLine(format,cikti[1],cikti[2],cikti[3]);
        }
        cikti.Close();
        baglanti.Close();
    }
}
```

Bu program ogrenci tablosundaki id hariç tüm verileri bir tablo şeklinde konsol ekranına yazar. Bu programda Temel string işlemleri konusunda gördüğümüz yazı formatlamayı kullandık. Aynı konuda gördüğümüz String sınıfına ait PadRight() metodunu ise ekrana bir karakteri belirli sayıda yazdırmak amacıyla kullandık. Asıl konumuza dönecek olursak OleDbDataReader sınıfının static olmayan Read() metodu ilgili tablodaki aktif kayıtı bir birim öteler ve true döndürür. Eğer aktif kayıt ötelenemiyorsa false döndürür. OleDbDataReader sınıfında tanımlı indeksleyici sayesinde aktif kayıttaki farklı alanlar tutulabilir. Bu indesleyici object türünde nesneler döndürür. Son olarak OleDbConnection ve OleDbDataReader nesneleriyle işimiz bittiğinde ilgili nesnenin Close() metoduyla sistemin ilgili tablo/bağlantı için ayırdığı sistem kaynaklarını işlemciye iade etmemiz gerekiyor. İstersek OleDbDataReader sınıfına ait indeksleyiciyi şöyle de kullanabilirdik:

```
Console.WriteLine(format,cikti["ad"],cikti["soyad"],cikti["not"]);
```

Gördüğünüz gibi indeksleyiciye bir indeks numarası vermektense direkt olarak sütunun adını da verebiliyoruz. Aynı programdaki while döngüsünü şöyle de kurabilirdik.

```
while(cikti.Read())
{
    Console.WriteLine(format,cikti.GetString(1),cikti.GetString(2),cikti.GetI
}
```

Burada klasik indeksleme yönteminin aksine türler belirtilmiştir. İndeksleyici geriye object nesnesi döndürüyordu. Eğer veri tabanındaki sütunların tipini biliyorsak bu şekilde bir kullanım tür dönüşümü olmadığından dolayı işlemler daha hızlı gerçekleşeceği için tavsiye edilir. Microsoft Office Access 2003 sürümündeki veri tipleri, C# karşılıkları ve bunları programa geçirmek için gerekli OleDbDataReader metotları şöyledir:

Access	C#	İlgili metot
Tamsayı	short	GetInt16()
Uzun Tamsayı	int	GetInt32()
Bayt	byte	GetByte()
Çift	double	GetDouble()
Tek	float	GetFloat()
Ondalık	decimal	GetDecimal()
Metin	string	GetString()
Evet/Hayır	bool	GetBoolean()
Tarih/Saat	DateTime	GetDateTime()

OleDbDataAdapter, DataSet, DataTable, DataRow ve DataColumn sınıfları

Artık şimdiye kadar öğrendiğimiz sınıflarla bir veri tabanından veri çekip veri tabanına veri kaydedebiliyoruz. Zaten bir veri tabanıyla yapabileceğimiz işlemler bu kadar. Ancak birazdan göreceğimiz OleDbDataAdapter, DataSet, DataTable, DataRow ve DataColumn sınıfları bu işlemleri yaparken bize daha fazla seçenek sunuyor. Ayrıca bu sınıflar bize offline çalışmanın kapılarını açıyor. Yani öncelikle veri tabanından bir veri çekiyoruz, bu veriyi kendi makinamıza aktarıyoruz, bu işlemden sonra veri tabanıyla bağlantımızın kesilmesi programımızın işleyişine engel değil. Sonra çektiğimiz veri üzerinde istediğimiz oynamaları yapıyoruz. Sonra değiştirilmiş verileri tekrar veri tabanına yazıyoruz. Bu şekilde veri tabanıyla işlemlerimiz, veri tabanının bulunduğu bilgisayarla programın bulunduğu bilgisayar arasında fazla git-gel olmadığı için daha hızlı gerçekleşiyor. Şimdi offline çalışmayla ilgili sınıfları önce kısaca inceleyelim:
OleDbDataAdapter OLEDB protokolünü destekleyen veri tabanlarından veri çekmek ve değiştirilmiş verileri aynı veri tabanına tekrar yazmak için kullanılır. Offline çalışmayla ilgili sınıflar içinde veri tabanıyla fiziksel olarak iletişimde olan tek sınıftır. OleDbDataAdapter sınıfının çektiği veri tek bir veri olabileceği gibi bir ya da daha fazla tablo da olabilir. Hatta istersek OleDbDataAdapter sınıfıyla bir veri tabanının tamamını da programımıza aktarabiliriz. OleDbDataAdapter sınıfı System.Data.OleDb isim alanındadır. System.Data.SqlClient isim alanındaki SqlDataAdapter sınıfı ise OLEDB'siz bağlanılan MSSQL veri tabanları için aynı şeyi yapar. OleDbDataAdapter ile SqlDataAdapter sınıflarının arayüzleri aynıdır. Dolayısıyla birazdan OleDbDataAdapter sınıfını incelerken aynı zamanda SqlDataAdapter sınıfını da incelemiş olacağız.
DataSet OleDbDataAdapter sınıfının veri tabanından çektiği verileri programda offline olarak tutmaya yarar. System.Data isim alanındadır.

DataTable Bir DataSet'teki bir tabloyu temsil eder. System.Data isim alanındadır.
DataRow Bir tablodaki tek bir satırı (kaydı) temsil eder. System.Data isim alanındadır.
DataColumn Bir tablodaki tek bir sütunu temsil eder. System.Data isim alanındadır.

Veri tabanından veri çekmek

Öncelikle bir OleDbDataAdapter nesnesi oluşturmalıyız:

```
string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb";
OleDbConnection baglanti=new OleDbConnection(kaynak);
baglanti.Open();
string sorgu="SELECT * FROM TabloAdı";
OleDbDataAdapter odda=new OleDbDataAdapter(sorgu,baglanti);
```

Sonra OleDbDataAdapter sınıfının Fill() metoduyla ilgili OleDbDataAdapter nesnesinin tuttuğu verileri bir DataSet nesnesine aktarmalıyız.

```
DataSet ds=new DataSet();
odda.Fill(ds); //Bu satırla odda nesnesindeki veriler ds nesnesine atandı.
```

İstersek OleDbDataAdapter nesnesinden DataSet nesnesine bir tablo aktarırken tablonun adını belirleyebiliriz.

```
DataSet ds=new DataSet();
odda.Fill(ds,"TabloAdı");
```

DataSet sınıfının Tables özelliği ilgili DataSet nesnesindeki tabloları bir DataTable koleksiyonu olarak döndürür.

```
DataSet ds=new DataSet();
odda.Fill(ds,"TabloAdı");
DataTable dt=ds.Tables["TabloAdı"];
//veya
DataTable dt2=ds.Tables[0];
//veya
DataTableCollection dtc=ds.Tables; //DataTableCollection bir koleksiyon sını
```

DataTable sınıfının Columns özelliği ilgili DataTable nesnesindeki sütunları bir DataColumn koleksiyonu olarak döndürür.

```
DataTable dt=ds.Tables["TabloAdı"];
DataColumn dc=dt.Columns["SütunAdı"];
//veya
DataColumn dc2=dt.Columns[2];
//veya
DataColumnCollection dcc=dt.Columns; //DataColumnCollection koleksiyon sınıf
```

DataTable sınıfının Rows özelliği ilgili DataTable nesnesindeki satırları bir DataRow koleksiyonu olarak döndürür.

```
DataTable dt=ds.Tables["TabloAdı"];
DataRow dr=dt.Rows[1];
//veya
DataRowCollection drc=dt.Rows; //DataRowCollection koleksiyon sınıfı System.
```

DataRow sınıfının ItemArray özelliği ilgili satırdaki verileri bir object dizisi olarak tutar. Yani

```
object[] o=dr.ItemArray;
```

Ayrıca bir DataRow nesnesindeki farklı sütunlara indeksleyici ile de erişilebilir. Örnek

```
object o1=dr[0]; //veya
object o2=dr["SütunAdı"];
```

Yine bu indeksleyiciler de object nesneleri döndürür.
NOT: DataTableCollection, DataRowCollection ve DataColumnCollection sınıflarının Count özellikleri bir datasetteki tablo sayısının ve bir tablodaki satır ve sütun sayısının bulunması amacıyla kullanılabilir.

Bir sütunun özelliklerini değiştirmek ve edinmek

DataColumn sınıfının çeşitli static olmayan özellikleri vardır:
bool AllowDBNull Sütunun boş değer kabul edip etmeyeceği belirtilir.
bool AutoIncrement Eklenen yeni kayıtlarda ilgili sütundaki verinin otomatik artıp artmayacağı belirtilir.
long AutoIncrementSeed Otomatik artacak değerin başlangıç değeri
long AutoIncrementStep Otomatik artımın kaçar kaçar olacağı belirtilir.
string Caption Sütunun ismi.
Type DataType Sütunun veri tipi.
object DefaultValue Sütundaki her hücrenin varsayılan değeri. (boş bırakıldığında)
int MaxLength Sütundaki her hücredeki verinin maksimum karakter sayısı.

int Ordinal Sütunun tabloda kaçıncı sırada olduğunu verir. (salt okunur)
bool Unique Sütunda bir verinin tekrarlanıp tekrarlanamayacağı. Tekrarlanabiliyorsa false, tekrarlanamıyorsa true.
DataTable Table Sütunun hangi tabloya ait olduğu (salt okunur)

Veri tabanını güncellemek

Bunu bir örnek üzerinde anlatmayı uygun buluyorum. Öncelikle kaynak kodumuzla aynı klasörde dosya.mdb isimli bir Access dosyası oluşturun. Bu dosyada tablo isimli bir tablo oluşturun. Tablo şöyle olsun:

id	ad	soyad
1	bekir	oflaz
2	mehmet	kaplan

id alanı Sayı (Uzun Tamsayı) diğer alanlar ise Metin tipinde olsun. Tabloda birincil anahtar olmasın. Şimdi programımızı yazmaya başlayabiliriz.

```
using System.Data.OleDb;
using System.Data;
class vt
{
    static void Main()
    {
        string kaynak="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dosya.mdb"
        OleDbConnection baglanti=new OleDbConnection(kaynak);
        string sorgu="SELECT * FROM tablo";
        OleDbDataAdapter odda=new OleDbDataAdapter(sorgu,baglanti);
        DataSet ds=new DataSet();
        odda.Fill(ds,"tablo");
        DataRow r=ds.Tables["tablo"].NewRow();
        r["id"]=21;
        r["ad"]="hatice";
        r["soyad"]="özdoğan";
        odda.InsertCommand=new OleDbCommand("INSERT INTO tablo (id,ad,soyad) v
        odda.UpdateCommand=new OleDbCommand("UPDATE tablo SET id=?, ad=?, soya
        odda.DeleteCommand=new OleDbCommand("DELETE FROM tablo WHERE id=?",bag
        odda.InsertCommand.Parameters.Add("@id",OleDbType.Integer,2,"id");
        odda.InsertCommand.Parameters.Add("@ad",OleDbType.VarChar,10,"ad");
        odda.InsertCommand.Parameters.Add("@soyad",OleDbType.VarChar,10,"soyad
        odda.UpdateCommand.Parameters.Add("@id",OleDbType.Integer,2,"id");
        odda.UpdateCommand.Parameters.Add("@ad",OleDbType.VarChar,10,"ad");
        odda.UpdateCommand.Parameters.Add("@soyad",OleDbType.VarChar,10,"soyad
        odda.UpdateCommand.Parameters.Add("@oldid",OleDbType.Integer,2,"id").S
        odda.DeleteCommand.Parameters.Add("@id",OleDbType.Integer,2,"id").Sour
        ds.Tables["tablo"].Rows.Add(r);
        ds.Tables["tablo"].Rows[0]["id"]=98;
        ds.Tables["tablo"].Rows[1].Delete();
        odda.Update(ds,"tablo");
    }
}
```

Bu program veri tabanında ekleme, değiştirme ve silme yaptı. Access dosyasındaki tablomuzun yeni hâlinin şöyle olması gerekiyor:

id	ad	soyad
98	bekir	oflaz
21	hatice	özdoğan

Programdaki önemli satırları teker teker inceleyecek olursak;

```
DataRow r=ds.Tables["Uyeler"].NewRow();
```

Burada ds DataSet nesnesindeki tablo tablosundaki düzene uygun bir DataRow nesnesi oluşturduk.

```
odda.InsertCommand=new OleDbCommand("INSERT INTO tablo (id,ad,soyad) values
odda.UpdateCommand=new OleDbCommand("UPDATE tablo SET id=?, ad=?, soyad=? WH
odda.DeleteCommand=new OleDbCommand("DELETE FROM tablo WHERE id=?",baglanti)
```

Bu üç satırda ise OleDbDataAdapter sınıfının ilk değer atanmamış ve tipi OleDbCommand olan üç özelliğine değer atadık. SQL cümlelerindeki ? işareti oraya bir parametre geleceğini belirtiyor. Burada OleDbDataAdapter sınıfının Update() metodu kullanıldığında aslında hangi komutların çalıştırılacağını belirledik. Bu satırlardan bazılarını yazmayabilirdik. Bu durumda yalnızca değer atanan özelliklerin komutları çalışırdı.

```
odda.InsertCommand.Parameters.Add("@id",OleDbType.Integer,2,"id");
odda.InsertCommand.Parameters.Add("@ad",OleDbType.VarChar,10,"ad");
odda.InsertCommand.Parameters.Add("@soyad",OleDbType.VarChar,10,"soyad");
odda.UpdateCommand.Parameters.Add("@id",OleDbType.Integer,2,"id");
odda.UpdateCommand.Parameters.Add("@ad",OleDbType.VarChar,10,"ad");
odda.UpdateCommand.Parameters.Add("@soyad",OleDbType.VarChar,10,"soyad");
odda.UpdateCommand.Parameters.Add("@oldid",OleDbType.Integer,2,"id").SourceV
odda.DeleteCommand.Parameters.Add("@id",OleDbType.Integer,2,"id").SourceVers
```

Burada ? işaretleri yerine ne geleceğini belirledik. Bunların eklenme sırası önemlidir. İlk ? yerine ilk eklenen gelir. Veri tiplerini System.Data.OleDb isim alanındaki OleDbType enumuyla belirledik. DataRowVersion enumu ise System.Data isim alanındadır.

```
ds.Tables["tablo"].Rows.Add(r);
ds.Tables["tablo"].Rows[0]["id"]=98;
ds.Tables["tablo"].Rows[1].Delete();
```

Bu üç satırda ise sırasıyla önceki oluşturduğumuz r satırını ds datasetindeki tablo tablosuna ekledik. Sonra 0. kaydın id sütunundaki değeri 98 yaptık. Sonra 1. kaydı sildik.

```
odda.Update(ds,"tablo");
```

Son olarak yaptığımız offline değişikliklerin veri tabanında da yapılmasını sağladık. Offline çalışmada bağlantının Open() ve Close() metotlarıyla açılıp kapanmasına gerek yoktur.

Veri sağlayıcı bağımsız erişim

Veri tabanlarına erişim veri sağlayıcılarla olur. Örneğin OLEDB bir veri sağlayıcıdır. OLEDB ile hemen hemen bütün veri tabanlarına bağlanabiliriz. Ancak MSSQL Server veya Oracle Server'ın kendine has veri sağlayıcıları da vardır. Bunlar veri tabanına göre optimize edildiği için veri tabanına erişimde ciddi bir hız kazancı sağlarlar. .Net Framework kütüphanesinde OLEDB protokolü için sınıflar yer almakla birlikte bu veri tabanlarına erişim için özelleşmiş sınıflar da mevcuttur. Şimdi karşımıza iki seçenek çıkıyor. Ya OLEDB ile hızdan ödün verip veri tabanı bağımsız erişim sağlayacağız ya da her veri tabanı için özelleşmiş sınıfları kullanacağız. Verilerimizin tutulduğu veri tabanı değiştiğinde bizim de programımızın kaynak kodunu tekrar değiştirip tekrar derleyip tekrar dağıtmamız gerekecek. Peki hem hızdan ödün vermeden hem de veri sağlayıcı bağımsız erişim mümkün olabilir mi? Evet, mümkün olabilir. Bunu C# geliştiricileri fabrika sınıf modeliyle başarmışlardır. .Net kütüphanesindeki fabrika sınıfları hakkında bilgiyi aşağıdaki programı yazarak bulabilirsiniz:

```
using System;
using System.Data;
using System.Data.Common;
class c
{
    static void Main()
    {
        DataTable saglayicilar=DbProviderFactories.GetFactoryClasses();
        foreach(DataRow satir in saglayicilar.Rows)
        {
            for(int i=0;i<saglayicilar.Columns.Count-1;i++)
                Console.WriteLine(satir[i].ToString());
            Console.WriteLine("".PadRight(15,'-'));
        }
    }
}
```

DbProviderFactories sınıfı System.Data.Common isim alanındadır. DbProviderFactories sınıfının static GetFactoryClasses() metodu, içinde .Net Framework kütüphanesindeki fabrika sınıflar hakkında bilgi olan bir tablo döndürür. Tablodaki her satır farklı bir fabrika sınıf içindir. Tablodaki her sütun ise ilgili fabrika sınıf hakkında farklı kategorideki bilgiler içindir. Tablodaki son sütun bizim için biraz fazla karmaşık bilgiler içerdiği için bu sütunu ekrana yazma gereksinimi görmedim. Bu programın bendeki çıktısı şöyle oldu:

```
Odbc Data Provider
.Net Framework Data Provider for Odbc
System.Data.Odbc
---------------
```

```
OleDb Data Provider
.Net Framework Data Provider for OleDb
System.Data.OleDb
---------------
OracleClient Data Provider
.Net Framework Data Provider for Oracle
System.Data.OracleClient
---------------
SqlClient Data Provider
.Net Framework Data Provider for SqlServer
System.Data.SqlClient
---------------
```

Bu ekran sizde farklı olabilir. Herhangi bir fabrika nesnesi oluşturmak için:

```
DbProviderFactory fabrika=DbProviderFactories.GetFactory("System.Data.SqlCli
```

Yani bir fabrika oluşturmak için az önceki tabloda karşımıza çıkan üçüncü sütundaki yazıyı kullanıyoruz. DbProviderFactory sınıfı System.Data.Common isim alanındadır. Konunun geri kalan kısmını bir örnek üzerinde anlatmayı uygun buluyorum:

```
using System.Data.Common;
using System;
class Fabrikalar
{
    static void Main()
    {
        DbProviderFactory fabrika=DbProviderFactories.GetFactory("System.Data.
        DbConnection baglanti=fabrika.CreateConnection();
        baglanti.ConnectionString="server=Sunucu;uid=Kullanici;pwd=sifre;datab
        baglanti.Open();
        DbCommand komut=fabrika.CreateCommand();
        komut.Connection=baglanti;
        komut.CommandText="SELECT * FROM TABLO";
        DbDataReader okuyucu=komut.ExecuteReader();
        while(okuyucu.Read())
            Console.WriteLine(okuyucu[1].ToString()+" "+okuyucu[2].ToString());
    }
}
```

Buradaki DbConnection, DbCommand ve DbDataReader sınıfları yine System.Data.Common isim alanındadır. Şimdiye kadar gördüğümüz veri tabanıyla ilgili sınıfların çoğunun System.Data.Common isim alanında erişim sağlayıcı bağımsız versiyonları vardır. Yine bu sınıfların arayüzleri aynıdır. Burada yaptığımız aslında veri tabanına özgü sınıflar kullanmak yerine hangi veri tabanına bağlanılacağını bir stringle belirlemekti. Böylelikle öncelikle kodlarla veri tabanından veri tabanının hangi veri tabanı olduğu bilgisi alınır. Sonra GetFactory() metodunun parametresine uygun string alınır. İşte bu sayede de veri sağlayıcıdan bağımsız işlemler yapılmış olur. Çünkü hangi veri tabanına bağlanılacağı derleme zamanında değil, programın karşılaşağı durumlara göre çalışma zamanında belirlenir.

Bu kitabın diğer sayfaları