

ADO.Net

Veri tabanı işlemleri için kullanacağımız yapıdır. Veri çekme, ekleme, güncelleme ve silme işlemlerini yapabilmemize olanak sağlar.

Veri Çekme İşlemi için aşağıda ki adımlar gerçekleştirilir.

Bağlantıyı oluştur.

Sorguyu oluştur.

Bağlantıyı aç.

Sorguyu çalıştır. (Veriyi elde et veya gönder)

Bağlantıyı kapat.

Ado.net 'in çalışabilmesi için Sql Server Managament Studio 'nun açık olmasına gerek yoktur. Yalnızca Sql girişinin (Servisinin çalışır durumda olması) açık olması yeterlidir.

Veritabanı Bağlantı Yöntemleri

SqlConnection

SqlConnection nesnesi veritabanı işlemlerimizi gerçekleştirmek için kullanacağımız nesnemizdir. System.Data.SqlClient kütüphanesi altında bulunur. Görevi, bizim Sql sorgularımızı çalıştırmak için gereken veritabanı bağlantısını kurmaktır.

Temel görevi, görsel olarak düşündüğümüzde bize MsSql ekranındaki Query sayfasını açmaktır. Bizden beklediği Server Name, Database Name, Username, Password bilgileri ile Sql servisine bağlanarak sorguları çalıştırabileceğimiz bağlantıyı oluşturur.

Sorguların çalışması için MsSQL Server'in açık olmasına gerek yoktur. Sql'e girişin yani servisin açık olması yeterlidir.

SqlConnection; Arkatarafta temel mantıkta yapılan işlem, Sql Management Studio'yu açar, giriş yapar, veritabanını seçer, new query ekranı açar.

Kütüphaneler

System.Data.Common

Veriye erişim kaynaklarında kullanılan sistemlere yönelik bir yapıdır.

System.Data.Odbc

Open Database Connectivity mantığına uygun yapıları içerir. Open Database Connectivity standartlarına uygun yapıları içerir.

System.Data.Sql ve System.Data.SqlClient

Sql'e özelleşmiş yapılardır.

System.Data.OleDb

Her türlü veritabanına erişmek için kullanılan ana yapıdır. Sql Client bu yapının özelleşmiş halidir. OleDb'de Connection nesnesi bulunur. Özelleşmiş yapılar (SqlClient, OracleClient, MySqlClient gibi) OleDb'den türemiştir. Burda OracleClient bulunmadığından, OracleClient kütüphanesini internetten bulup projeye dahil ettiğimizde diğer veritabanlarının da kullanabiliriz.

System.Data.ProviderBase

Kendi bağlantımı oluşturmam için bana yapılar sunar. Ürettiğimiz bir database yapısı olursa bu namespace'den tanımlama yapılır.

System.Data.SqlTypes

Sqlde tanımlı olan, c sharp da tanımlı olmayan yapıları c sharp tarafında birebir kullanmak için tanımlı olan yapıdır. (money gibi.)

ConnectionString Yazmak

1.Yöntem : ConnectionString'i Constructor'da yazmak

```
SqlConnection baglan = new SqlConnection("Server=<ServerAdi>; Database=<VeritabaniAdi>;  
UID=<KullaniciAdi>; PWD=<Sifre>");
```

2.Yöntem : ConnectionString'i Property'de yazmak

```
SqlConnection baglan = new SqlConnection();
```

```
baglan.ConnectionString = "Server=<ServerAdi>; Database=<VeritabaniAdi>; UID=<KullaniciAdi>; PWD=<Sifre>";
```

3.Yöntem : ConnectionString'i Class'da yazmak

```
public static class BaglantiCumlesi  
{  
    private static string _connectionString;  
    public static string ConnectionString  
    {  
        get  
        {  
            return "Server=<ServerAdi>; Database=<VeritabaniAdi>; UID=<KullaniciAdi>; PWD=<Sifre>";  
        }  
    }  
}
```

4.Yöntem: ConnectionString'i Global'de Yazmak

```
SqlConnection baglan = new SqlConnection("Server=<ServerAdi>; Database=<VeritabaniAdi>;  
UID=<KullaniciAdi>; PWD=<Sifre>");
```

```
private void Form1_Load(object sender, EventArgs e)  
{
```

```
}
```

AppConfig dosyası içerisinde ConnectionString Kullanmak

Namespace'lere System.Configuration dosyasını eklememiz gerekir. System.Configuration kütüphanesi, bizim configuration dosyamızı yönetmemizi sağlar. Kütüphaneyi referanslara eklemek için; References Sağ Tık > Add Reference à Arama alanına Configuration yazdığımızda çıkan System.Configuration'ı eklememiz gerekmektedir.

ConnectionString'i ConfigurationFile'dan Çekmek

```
<!-- CONFIGURATION FILE (App.Config / Web.config) -->
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="Baglan" connectionString="Server=localhost; Database=Northwind; UID=sa; PWD=123"
    providerName="System.Data.SqlClient" />
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
</configuration>
//ConnectionString
SqlConnection baglan = new
SqlConnection(ConfigurationManager.ConnectionStrings["Baglan"].ConnectionString);
```

CONNECTED MİMARİ

Ado bağlantılıdır. Bu yapıda SQL'e bağlantıyı kendimiz yönetiriz. Bağlantıyı açmak ve kapatmak yazılımcı tarafından yönetilir.

SqlConnection Bağlantıyı Yönetmek

Bağlantıyı Açmak ve Kapatmak

```
if (baglan.State == ConnectionState.Closed) //Bağlantı kapalıysa, sonuç True Döner..
{
    baglan.Open(); //Bağlantıyı Açan metod.
    MessageBox.Show("Bağlantı Açıldı!");
}
else
{
    baglan.Close(); //Bağlantıyı Kapanan metod    baglan.Dispose(); //Bağlantıyı RAM'den düşürür.
    MessageBox.Show("Bağlantı Kapandı");
}
```

ConnectionState: Bize bağlantının durumunu true ve false olarak geri dönecektir.

SqlCommand

Şu ana kadar sadece veritabanı bağlantısını oluşturduk. Bağlantıları açtık veya kapattık. Ama henüz veritabanı ile ilgili işlem yapmadık. Artık bağlantılarımızı açtıktan sonra Sql sorgularını çalıştırmaya başlayabiliriz. Bu işlem için SqlCommand nesnesini kullanabiliriz.

SQL taraflı olarak bir sorgu çalıştırmak isterseniz referans alacağınız nesne "SqlCommand" nesnesidir.

SqlCommand Property(Özellikleri)

Connection

Çalıştırılacak sql cümlesinin, hangi veritabanı bağlantısını kullanacağını gösterir. CommandType : Çalıştırılacak sql cümlesinin tipini belirler. Query, StoredProcedure, TableDirect değerlerini alır.

Query

Sql cümlesinin çıplak olarak kullanıldığı özellik

StoredProcedure

Sql cümlesinin Stored Procedure olarak kullanıldığı özellik

TableDirect

Sql cümlesi yerine direk olarak tablo ismi verilerek kullanıldığı özellik. (Birden fazla tablo kullanılacak ise, tablo isimleri , işareti ile ayrılır.)

CommandText

CommandType özelliği göz önünde bulundurularak sql cümlesi, stored procedure veya tablo isminin yazıldığı özellik.

CommandTimeout

Sql cümlesinin çalıştırılması ve cevap dönme süresi için bekleme zamanını belirtir. Varsayılan değeri 30'dur. Süre saniye cinsinden verilir.

Transaction

Çalıştırılan sql cümlesine, oluşturulan transaction nesnesi bağlanarak, ADO.NET düzeyinde kontrol yapılması sağlanabilir. Bu uygulamanın bize sağladığı fayda, çalıştırılan sql cümlesinin, çalıştırılması sonucu oluşan hatalarda geriye dönüş yapılabilmesine imkan vermesidir. Çoğunlukla, Insert/Update/Delete işlemlerinde kullanılır.

Parameter

SqlCommand nesnesi aracılığı ile çalıştırılan Sorgulara ve Stored Procedure'lere tanımlanan parametrelere değer göndermek için kullanılır.

1.Yöntem : Query String

```
SqlCommand cmd = new SqlCommand("Select * from Categories", baglan);
```

2.Yöntem : StoredProcedure

```
SqlCommand cmd = new SqlCommand("StoredProcedureAdi", baglan);  
cmd.CommandType = CommandType.StoredProcedure;
```

Yapı

```
SqlConnection baglan = new SqlConnection("Server=.; Database=Northwind; UID=sa; PWD=123"); //Bağlantı nesnemiz
```

```
SqlCommand cmd = new SqlCommand("Insert Into Categories (CategoryName, Description) VALUES ('Yeni Kategori', 'Açıklaması')", baglan); //baglan nesnesini kullanarak kategoriler tablosuna insert yapan sorguyu verdik.
```

```
if (baglan.State == ConnectionState.Closed) baglan.Open(); //Bağlantıyı açar.
```

```
int etkilenen = cmd.ExecuteNonQuery(); //Etkilenen kayıt sayısını döner
```

```
baglan.Close(); //Bağlantıyı kapattık baglan.Dispose(); //RAM'den düşürdük
```

SqlCommand Metotları

ExecuteNonQuery

Etkilenen satır sayısını geriye döndürür. ExecuteScalar : Result set'in birinci sütununun birinci satırını geri döndürür.

ResultSet

Select * from dediğimizde dönen tablodur. ExecuteReader : ResultSet'in tamamını geriye döndürür.

Execute metotlarından bana dönen değeri kullanabilirim. Örneğin ExecuteNonQuery() bana etkilenen satır sayısını döndürür. Ben bu sayıyı kontrol ederek ekleme işleminin gerçekleşip gerçekleşmediğine bakabilirim.

Execute Nesnesinden Dönen Değerin Kontrolü

//1.Yöntem

```
int etkilenenSatirSayisi = cmd.ExecuteNonQuery(); //Etkilenen satır sayısını elde ederiz.
```

//2.Yöntem

```
bool sonuc = cmd.ExecuteNonQuery() > 0; //Etkilenen satır sayısı 0'dan büyükse true değer döner. Buda işlemin başarıyla gerçekleştiğini gösterir.
```

Insert İşlemi Yapmak

//Dışardan girilen kategori adı ve açıklamasına insert yapan kodlar

```
SqlConnection baglan = new SqlConnection("Server=.; Database=Northwind; UID=sa; PWD=123");
```

```
SqlCommand cmd = new SqlCommand("Insert Into Categories (CategoryName, Description) VALUES ('" + txtKategoriAdi.Text + "', '" + txtAciklama.Text + "')", baglan);
```

```
if (baglan.State == ConnectionState.Closed) baglan.Open(); bool sonuc = cmd.ExecuteNonQuery() > 0;
```

```
//Etkilenen satır sayısı 0'dan büyükse değişkene true değer atar.
```

```
if (sonuc) //Eğer etkilenen satır sayısı 0'dan büyükse true döner
    MessageBox.Show("Kategori başarıyla kayıt edildi.");
else
    MessageBox.Show("Kategori kayıt edilirken bir hata oluştu.");

baglan.Close();
baglan.Dispose();
```

DISCONNECTED MİMARİ

Disconnected Mimari; sql sorgularımızı çalıştırmak için oluşturduğumuz bağlantıyı açma/kapama işlemini üstlenir. Disconnected mimaride bağlantıları yapının kendisi yönetir.

Bunun dışında disconnected mimaride veri çekme hızı connected mimariden daha yüksektir. Connected mimari ile dönen ResultSet'i satır satır okumak durumundayız. Ama disconnected mimari, veri okuma işlemleri için kullanılan hazır yapıları kullanarak daha hızlı bir şekilde verileri çeker.

Veri çekme işlemi arasında ki hız farkını Entity Framework'de inceleyeceğiz. Şimdi Disconnected Mimaride kullanılan yapıları inceleyelim.

SqlDataAdapter

Veritabanından çekilen veriyi DataTable'ımıza doldurmak veya DataTable'ımızda güncellenerek cachelenmiş verilerinizi veri kaynağınızda da güncellemek için kullanılır.

- SqlDataAdapter Disconnected Data ile çalışmak için tasarlanmıştır.
- SqlDataAdapter ve DataSet arasında direkt olarak bir bağlantı yoktur.

DataAdapter ile DataSet'e veri doldurulurken database bağlantısı kapalı ise açılır, DataSet doldurulur ve işlem bittiği anda otomatik olarak kapatılır. Eğer bağlantı halihazırda açıksa herhangi bir hata vermeden işlemi yapar ve bağlantıyı kapatarak işlemini sonlandırır. Böylece bizlerin de fazla detayla uğraşmadan hızlıca işlemlerimizi yapmamızı sağlar.

Constructor'da iki parametre ister. Birinci parametre Sql Sorgusu, ikinci parametre ise SqlConnection nesnesidir. Son olarak DataSet'e veriyi doldururken de Fill() methodu kullanılır.

DataTable

Veri Kaynaklarından elde ettiğimiz bilgileri, veritabanından bağımsız olarak tablo formatında tutmamıza olanak sağlayan yapıdır. ADO.NET .Net Framework 2 ile birlikte veritabanından verileri alarak Dataset'e ihtiyaç duymadan direkt Datatable'e aktarabilme gibi bir yetenek kazandırılmıştır.

Disconnected Mimari kullanarak Northwind veritabanımızda bulunan ürünleri projemizde bulunan DataGridView kontrolüne aktaralım. Connected veya Disconnected mimari kullanırken SqlConnection nesnesi tanımlamalıyız. Sql Connection bizim sorgularımızın hangi server ve hangi veritabanı üzerinde hangi kullanıcı yetkisi ile çalışacağını belirler.

Örnek Uygulama

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

```
{
```

```
//Aşağıdaki kod satırında SqlDataAdapter nesnesini oluşturuyoruz. Çalışacak sql sorgusunu belirtiyoruz. Yazdığımız
```

```
sql sorgusunun hangi connection üzerinden çalışacağını belirliyoruz.
SqlDataAdapter dap = new SqlDataAdapter("Select * from Products", new SqlConnection("Server=ALP1-
BILGISAYAR\\EMPATI; Database=Northwind; UID=sa; PWD=123"));
DataTable dt = new DataTable();//Yeni bir DataTable nesnesi oluşturuyoruz.
dap.Fill(dt); //Veritabanından gelen ResultSet'i Oluşturulan DataTable nesnesine dolduruyoruz.
dataGridView1.DataSource = dt;//DataTable'ı veri kaynağı olarak kullanıyoruz.
}
```

DataSet

Veri kaynağından elde edilen verileri, veritabanı gibi bir modelde tutmak istediğimiz zaman bize yardımcı olan yapıdır. Veritabanımızda tablo yapıları bulunmaktadır. Tablo içerisinde ki veriler satır ve sütunlarda bulunur. Projelerimizde veritabanından veri çektiğimiz zaman, veritabanında bulunduğu verileri tablo halinde elde edebiliriz. DataSet nesnesi içerisinde birden fazla DataTable bulundurarak bizim için bir çok tabloyu tek bir çatı altında toplar. Yani DataSet içerisinde veritabanından bağımsız olarak, veritabanımızdan gelen birden fazla tablo bulunabilir. Dataset aracılığı ile veri tabanımızdaki verileri tabloda bulunduğu yapısı ile uygulamamıza aktarabiliriz. İşin güzel tarafı bu verileri dataset'e bir kez aktardıktan sonra veriler sunucunun hafızasında saklanacağından artık database ile ilgimiz kalmamakta. Sunucunun hafızasında bulunan bu verilerle görüntüleme, silme, güncelleme gibi işlemlerimizi kolaylıkla yapabiliriz.

Örnek Uygulama

```
public Form1()
{
    InitializeComponent();

    //Global Tanımlamalar DataSet ds = new DataSet(); //DataSet'e farklı eventlardan erişmek için Global
    tanımlama yapıyoruz.

    //Formun Load Eventi
    private void Form1_Load(object sender, EventArgs e)
    {
        SqlConnection baglan = new SqlConnection("Server=ALP1-BILGISAYAR\\EMPATI; Database=Northwind;
        UID=sa; PWD=123");

        //DataAdapter kullanarak 3 farklı DataTable'a veritabanındaki tablolarımızı dolduruyoruz.
        SqlDataAdapter dap = new SqlDataAdapter("Select * from Categories", baglan);
        DataTable dt = new DataTable();//Yeni bir DataTable nesnesi oluşturuyoruz.
        dap.Fill(dt);

        SqlDataAdapter dap2 = new SqlDataAdapter("Select * from Products", baglan); DataTable dt2 = new
        DataTable(); dap2.Fill(dt2);

        SqlDataAdapter dap3 = new SqlDataAdapter("Select * from Orders", baglan); DataTable dt3 = new
        DataTable("Tablom"); //Constructor'da tabloya bir isim veriyorum.      dap3.Fill(dt3);

        //Oluşturduğumuz DataSet nesnemizin içerisinde DataTable nesnelerimizi tablo olarak ekliyoruz.
        ds.Tables.Add(dt);
        ds.Tables.Add(dt2);
        ds.Tables.Add(dt3);
    }

    //Kategori listelemek için kullanılacak butonun click eventi
```

```
private void btnKategorileriListele_Click(object sender, EventArgs e)
{
    //DataSet içerisindeki tabloya ulaşmak için iki yöntem vardır.
    //--1.Yöntem : Sıra numarası ile
    dataGridView1.DataSource = ds.Tables[0]; //DataSet içerisinde bulunan DataTable'lar dan indexi 0 (dt
    Tablosu) olan
}

//Ürün listelemek için kullanılacak butonun click eventi
private void btnUrunleriListele_Click(object sender, EventArgs e)
{
    dataGridView1.DataSource = ds.Tables[1]; //DataSet içerisinde bulunan DataTable'lar dan indexi 0 (dt
    Tablosu) olan
}

//Sipariş listelemek için kullanılacak butonun click eventi
private void btnSiparisleriListele_Click(object sender, EventArgs e)
{
    dataGridView1.DataSource = ds.Tables["Tablom"]; //DataTable ismi ilede çağırabilirim.
}
```

DataColumn & DataRow

DataRow; DataTable'da bulunan kolon nesneleridir. DataTable'a kod ile yeni bir kolon ekleyeceğimiz zaman kullanabiliriz. DataRow; DataTable'da bulunan her bir satırdır. DataTable'a yeni bir satır eklemek istediğimizde veya DataTable'da bulunan bir satırı almak istediğimizde DataRow nesnesini kullanırız.

Aşağıdaki örnekte; YeniDoganBebekler isimli bir DataTable nesnesinde, doğan bebeklerin bilgileri tutulmaktadır. Kod ile yeni doğan bebeklerin bilgilerini ekleyip bu bebeklerden bir tanesini DataRow nesnesine aktarma işlemini gerçekleştireceğiz.

Örnek Uygulama

```
public Form5()
{
    InitializeComponent();
}

DataSet ds = new DataSet("Uyelik");

private void Form5_Load(object sender, EventArgs e)
{
    //Tablo olusturulur...
    DataTable dt_Uyeler = new DataTable("Uyeler");

    //Primary key kolonu olusturulur...
    DataColumn dc_UyeID = new DataColumn("UyeID");
    dc_UyeID.DataType = typeof(int);
    dc_UyeID.AllowDBNull = false;
    dc_UyeID.AutoIncrement = true;
    dc_UyeID.AutoIncrementSeed = 1;
```



```
dc_UyeID.AutoIncrementStep = 1;

//Diğer kolonlar oluşturulur..
DataColumn dc_UyeAdi = new DataColumn("UyeAdi");
dc_UyeAdi.DataType = typeof(string);
dc_UyeAdi.AllowDBNull = false;
dc_UyeAdi.MaxLength = 24;
dc_UyeAdi.Unique = true;

DataColumn dc_Telefon = new DataColumn("UyeTelefonu");
dc_Telefon.DataType = typeof(string); dc_Telefon.AllowDBNull = false;

//Oluşturulmuş olan kolonlar tabloya eklenir....
dt_Uyeler.Columns.AddRange(new DataColumn[] { dc_UyeID, dc_UyeAdi, dc_Telefon });

//Tablonun primary key kolonu atanır...
dt_Uyeler.PrimaryKey = new DataColumn[] { dt_Uyeler.Columns[0] };

//Tablo tamamen hazır olduğuna göre db içerisine (dataSet) eklenir...
ds.Tables.Add(dt_Uyeler);
}

private void btnUyeKaydet_Click(object sender, EventArgs e)
{
    //Eklenecek satır oluşturulur...
    DataRow dr_Eklenecek = ds.Tables[0].NewRow();
    dr_Eklenecek[1] = txtUyeAdi.Text;
    dr_Eklenecek[2] = txtUyeTelefonu.Text;

    //Satiri oluşturmamak yetmez, o satiri ilgili tabloya eklemeniz gerekmektedir...
    ds.Tables[0].Rows.Add(dr_Eklenecek);

    //Son olarak dataSet üzerinde yapılan tüm değişiklikler onaylanır...
    ds.AcceptChanges();

    dgvUyeler.DataSource = ds.Tables[0];
    txtUyeAdi.Clear();
    txtUyeTelefonu.Clear();
}
}
```

OBJECT RELATIONAL MAPPING

ORM, *Object Relational Mapping* anlamına gelmektedir. İlişkisel veritabanı (RDBMS) ile nesneye yönelik programlamanın (OOP) arasında bir tür köprü özelliği gören ve ilişkisel veritabanındaki bilgilerimizi yönetmek için, nesne modellerimizi kullandığımız bir tekniktir.

AVANTAJLARI VE DEZAVANTAJLARI NELERDİR?

Bir porje geliştirmeye başladığınızda ORM araçlarını kullanmaya karar vermeden önce aşağıdaki avantaj ve dezavantajları gözden geçirmeliyiz..

Avantajları

- Nesneye yönelik bir programlama metodu sunmaktadır.

- Yazılan kodun veritabanı ile bağımlılığı bulunmamaktadır. Her kütüphane kullanmak istediğimiz veritabanı ile çalışabilmektedir. (Oracle, SQL Server, MySQL vs)
- SQL yazmanıza gerek kalmadan çok kısa bir zamanda ve de çok daha az kod ile veritabanına bağlı bir uygulama yapabilirsiniz.
- ORM araçlarının çoğu Open Source (Açık kaynak kod) olarak lisanslandığından ücretsiz olarak kullanabilir, geliştirebilir ve geliştirilmesine destek verebilirsiniz.
- ORM araçları, programcılara bir çok kolaylık sağlıyor ve içinde barındırdığı ek desteklerle bir çok sık görülen sorunlara çözüm sunuyor. (polymorphism, caching, transaction vb.)
- Çok daha iyi test edilebilir kod yazmamızı sağlamaktadır.

Dezavantajları

- ORM araçlarında çeşitli performans sorunları olabilmektedir.
- Bilgi alış-verişi sırasında kontrolün yüzde yüz sizin elinizde olmaması (Üretilen SQL sorguları beklenenden farklı çalışabiliyor fakat ORM aracı üzerinde ileri seviye teknik bilgi sahibi olduğunuzda bir noktaya kadar bu işlemi kontrolünüz altına alabilirsiniz.)

Popüler ORM Araçları Nelerdir?

Orm'den bahsedipte popüler ORM araçlarından bahsetmemek olmaz tabiki. .Net için sektörde kullanılan en popüler ORM araçlarının başında Entity Framework ve nHibernate gelmektedir. Fakat bu araçlar dışında aşağıdaki listede olduğu gibi bir çok farklı ORM aracı bulunmaktadır. Bu araçlar; .Net Persistence, BBADDataObjects, DataObjects.NET, DotNorm, FastObjects.NET, Norm, OJB.NET olarak gösterilebilir. Bu ORM araçları dışında daha bir çok ORM aracı bulunmaktadır.

ORM ile Neler Yapılabilir?

ORM sayesinde CRUD başta olmak üzere SQL sorgularıyla yapılan birçok işlem SQL sorgusu kullanılmadan gerçekleştirilmektedir. T-SQL üzerinde (veya kullanılan veritabanına göre başka bir SQL dili..) yapılabilen hemen hemen her türlü işlem ORM mantığıyla gerçekleştirilebilmektedir. SQL üzerinde kullanabildiğiniz Aggregate Function, Transaction gibi yapıları destekleyebilirler.

ORM'nin Çalışma Mantığı

ORM'nin en önemli özelliği veritabanı bağımsız çalıştırabilmektir. Burada dikkat etmeniz gereken nokta ORM kütüphanelerini kullanırken bazı ORM araçları, bazı veritabanları ile bağımsız çalışma özelliğini kısıtlamaktadır. ORM'de tüm işlemler nesneler ve nesneler arası ilişkiler ve bu ilişkilerin UYGULAMA KATMANINDA tanımlanması sebebiyle veritabanından sadece TABLO ve tabloların alanlarının istenen formatta olması yeterlidir.

ORM'nin Avantajları Nelerdir?

En büyük avantajı Transaction Yönetimidir. Bir Transaction başlatıp başlattığımız transaction içerisinde birçok INSERT/UPDATE/DELETE işlemi gerçekleştirebiliriz. **ORM**, kullandığı veritabanının yansımını oluşturmaktadır. Yansıma üzerinde işlemler yapıldıktan sonra "Değişiklikleri Veritabanına Kaydet" komutu verilmeden bu işlemler localde gerçekleşmiş olacaktır. Kayıt işlemini gerçekleştirdiğimiz anda (Transaction tamamlandığında) veritabanına etkileyecektir.

Veritabanı üzerinde çeşitli işlemler yaptığımızda bu işlemler local üzerinde yapılacaktır. İşlemler local üzerinde gerçekleştirildiğinden veritabanına kaydet komutu verildiğinde tüm işlemler veritabanı üzerinde etkilenecektir. Bu yöntem her işlem yaptığımızda veritabanına bağlanma ve bağlantı kapatma gibi işlemlerden kurtulmamızı sağlamaktadır. Her seferinde bağlantı açılıp kapanmadığından dolayı performans artışı sağlanacaktır.

ORM ile Veritabanı Modelleme Zorunlu mu?

ORM'in bir başka avantajı da veritabanı modellemeyi uygulama tarafında yapabilmeye izin vermesidir. Tablolar arası bağlantıları veritabanı katmanında yapabileceğimiz gibi, uygulama katmanında da gerçekleştirebiliriz. Bunun

avantajı ise, veritabanının başka bir sunucuya (SQL Server'dan Oracle'a gibi) aktarımında tablolar arası ilişkileri yeni veritabanında da oluşturmak için zaman kaybetmeyi önlemektedir. Configuration dosyası üzerinde yapılacak ufak değişikliklerle farklı veritabanlarını kullanmak mümkün olmaktadır.

ORM'de Mapping Nedir?

Mapping, ORM'de veritabanı ile nesnelerimiz arasındaki bağı kuran yapımızdır. Hangi sınıfın hangi tabloyla bağlanacağını, bağlanan tablolarda hangi property'nin (özellik'in ya da değişken'in) tablonun hangi alanıyla bağlanacağını, tablonun özelliklerini (ID'sinin ne olduğu, ID'sinin autoincrement olup olmadığı vb.) bilgilerin tanımlandığı yapımızdır. Kimi ORM frameworkleri bu işlemi yazılımcının yapmasını istese de bir çok framework bu işlemleri kendisi gerçekleştirmekte, istediği taktirde yazılımcının mapping'de düzenleme yapmasına izin vermektedir.

ORM Aracı Seçiminde Dikkat Edilmesi Gerekenler

Bir uygulama geliştirirken kullanılması gereken ORM aracı seçimine dikkat etmeliyiz. ORM aracı seçerken aşağıdaki maddeleri göz önünde bulundurunuz.

Uygulama Ölçeği

Bir uygulama geliştirirken kullanılacak teknolojileri seçmek için uygulamanın büyüklüğü önemli bir kriterdir. Aşağıdaki bilgiler bu tercihi yaparken ORM bilginizin ve tecrubenizin olmaması veya çok az olması durumları göz önünde bulunarak yazılmıştır.

Küçük Ölçekli Uygulamalarda; basit bir ORM aracı tercih edebilir veya ORM aracı kullanmayabilirsiniz.

Büyük Ölçekli Uygulamalarda; bir ORM aracı kullanmanız tavsiye edilmektedir. ORM araçlarının sunduğu olanaklar geliştirme süresini pozitif yönde kısaltacağı gibi, kod satırlarını azaltacak ve daha kolay yönetebileceğiniz bir uygulama geliştirmenize imkan tanıyacaktır. Bu durumda sadece hangi ORM aracını seçmeniz gerektiği ile ilgili karar vermeniz gerekecektir.

Uygulama Performansı

ORM araçlarının kullanımında en büyük sorunların başında performans sorunları yer alıyor. ORM performans düşüklüğünün nedeni kontrolün elimizde olmadığından üretilen karmaşık SQL sorgularından kaynaklıdır. Karmaşık nesne modelleriyle çalışırken, veri saklama/silme ya da alma işlemi sırasında ORM aracı tarafından üretilen SQL sorguları her zaman performanslı değildir. Bunları çok iyi analiz edip, gereken değişiklikleri yapmamız gerekebilir. Bu değişiklikler için kullanılan ORM aracı üzerinde etkin bir bilgiye sahip olunması gerekmektedir.

Bir diğer performans etkenide ORM aracının yanlış kullanımıdır. Bunun en büyük örneği Lazy Loading ve Eager Loading gibi özelliklerin bilinçsiz kullanımı veya kullanılmamasıdır.

Her zaman ORM aracı ile sorunlarınızı çözemeyebilirsiniz. Bu gibi durumlarda ORM aracı ile birlikte ADO.Net sorgularıda kullanabilirsiniz. Tabi bu kullanımda cache ve nesne durumlarının sorun oluşturup oluşturmayacağını dikkatle kontrol etmeniz gerekmektedir.

Uygulama içinde ve ORM aracı ile ilgili yazdığınız kodların yanı sıra veritabanı tasarımıda performansı etkilemektedir. Doğru bir veritabanı tasarımı performansda pozitif artış gösterecektir. Veritabanı üzerindeki index yapıları ve bağlantı havuzunuz doğrudan performans ile ilgilidir.

Projenin Deadline Süreci

Projenin bitiş tarihi ORM aracı seçiminizi doğrudan etkileyebilecektir. Uygulamayı geliştirecek ekip içerisinde herhangi bir geliştiricinin ORM aracı hakkında belli bir tecrübe yoksa, bu projeniz için büyük bir risk taşımaktadır. Bilinçsiz kullanıldığında ORM araçları yarardan çok zarar getirebilirler. ORM araçları başlangıçtaki basit sorgular için kolay görünebilirler fakat ileri seviye sorgularda ve tekniklerde zorluklar çekebilirsiniz. Yeterli zamanınız ve tecrubeniz yoksa ORM araçlarını doğrudan bir projede kullanmanız tavsiye edilmez.

ORM Aracı Kullanımı ile İlgili Tavsiyeler

- Lazy Loading, Eager Loading gibi kavramları iyi tanıyın ve kullanmaya çalışın.
- Caching mekanizmasını iyi anlayın ve ihtiyacınıza göre değiştirin.
- Çok karışık ORM modelleri tasarlamaktan kaçının. Ne kadar tecrübeli olursanız olun ihtiyacınızdan daha karmaşık yapılar zorluklar çıkaracaktır.
- ORM araçlarının her türlü sorunlara çözüm olamayabileceğini kavrayın, yeri geldiğinde SQL sorguları kullanmaktan çekinmeyin.
- Veritabanı tasarımınızı en iyi şekilde hazırlayın. Doğru hazırlanmamış bir veritabanı hangi ORM aracı kullanılırsa kullanılsın kararlı çalışmayacaktır. Mümkünse veritabanınızı bir DBA (DataBase Administrator) tarafından kontrol ettirmenizdir.
- Veri listeleme işlemleri yaparken mümkün olduğu kadar pagination (sayfalama) kullanın böylelikle tüm veriyi bir kerede almamış olacaksınız. Tüm veriyi tek seferde çekmek performans kaybına neden olacaktır.
- Veritabanından listeleme yaparken sadece ihtiyacınız olan kolonları getirin.
- ORM aracının kullandığı mekanizmaları veya kullandığınız mekanizmaları kullanımınıza uygun olarak optimize edin.
- Kullanacağınız ORM aracı yöntemini doğru seçin.

Sonuç olarak ORM araçları proje geliştirirken büyük bir destek sağlamaktadır. Veritabanı işlemlerinin karmaşıklığını gizler. Yanlış kullanıldığında tam bir kabus olabilecektir. Bu sebepten kullandığınız ORM aracı ne olursa olsun detaylarını öğrenin ve mümkünse ORM aracının geliştiricileri tarafından paylaşılan dökümantasyonları okuyun.

ENTITY FRAMEWORK

Entity Framework (Varlık Kütüphanesi) Microsoft tarafından geliştirilen ve yazılım geliştiricilerin katı sql sorguları yazmalarını ortadan kaldırarak bir ORM (Object Relational Mapping) imkanı sağlayan framework'tür. ORM'in ise ilişkisel veritabanı yönetim sistemlerine direkt olarak müdahale yerine nesneler aracılığı ile müdahale edilmesini sağlayan bir köprü olduğunu ORM başlığı altında belirtmiştik.

Entity framework'ün temel amacı uygulama geliştiricinin data işlemleri içerisinde kaybolmadan uygulama tarafına odaklanmasını sağlamaktır. Çok basit bir örnek olarak, klasik ADO.NET uygulamalarında bir bağlantının açılmasından ve kapatılmasından tamamen biz geliştiriciler sorumludur. Ancak entity framework kullandığınızda bu tür işlemleri entity framework kütüphanesi yönetmektedir. Sorgularınızı Linq To Entities formatında hazırlar ve Entity Framework aracılığı ile database üzerine iletirsiniz..

Piyasada bir çok ORM Framework'leri bulunmaktadır. Örnek olarak; DataObjects.Net, NHibernate, OpenAccess, SubSonic etc. Entity Framework gibi ORM kütüphaneleri bulunmaktadır.

Entity Framework; veritabanı katmanımızın uygulama katmanına yansımalarının alınması mantığı ile kullanılır. Entity Framework kütüphanesinin üç yaklaşımı bulunmaktadır. Bu yaklaşımlar;

- Database First
- Model First
- Code First

yaklaşımlardır. Bu yaklaşımları sırasıyla inceleyeceğiz.

Database First

Kullandığımız SQL veritabanı üzerinde hali hazırda tasarlanmış bir veritabanımız olduğunda, veritabanının yansımalarının (Yansıma : Veritabanındaki tabloların sınıflara ve tablolar içerisindeki kolonların ise sınıflar içerisindeki property'lere yazdırılması işlemidir.) alınması işlemidir. Yani tabloları Class olarak, kolonları ise property olarak yazdırır. Bizler yansıma alırken oluşturulan classları kullanarak veritabanı işlemlerimizi gerçekleştirebiliriz. Veritabanı tasarımı doğru yapıldığında yazdırma ve kullanma sırasında bir sıkıntı çıkartmayacaktır. Database First yaklaşımında yansıma alma işlemi T4 isimli bir sistem tarafından yazdırılmaktadır.

Database First yaklaşımı diğer yaklaşımların aksine hazır bir database üzerinden ilerlememizi sağlar. Hali hazırda bulunan database'in yansımaları alındıktan sonra eğer bir değişiklik yapılacaksa sql server üzerinden manual olarak yapmalıyız. Bu durumun projeye yansımaları içinde değişikliğin yansımada da geçerli olması için "update model from database" tıklamamız gerekli.

Model First

Eğer bir veritabanına sahip değilseniz, visual studio üzerinden classları kullanarak veritabanını oluşturabilirsiniz. Burada gerçekleştirdiğimiz değişiklikleri sql'e tanıtmak için "generate database from" seçeneğine tıklamamız gerekir. Bu yöntemde genellikle kod yazmak yerine design penceresi ile çalışırız. Model First yaklaşımında veritabanını classlar ile tasarladığınızda farklı veritabanı sistemleri ile çalışmanız gereken durumlarda connection konfigürasyonu yaparak veritabanınızın farklı sistemlerde oluşmasını sağlayabilirsiniz.

Code First

Veritabanınızı kod yazarak oluşturmak istediğinizde kullanacağınız yaklaşımdır. Bu yaklaşımda Model First gibi classları ve propertyleri geliştirici oluşturacaktır. Kullanmak için hazır bir veritabanına ihtiyacınız yoktur. Bu yaklaşımda Model First yaklaşımındaki gibi tüm classlar el ile kodlarla yazılmaktadır. Code First yaklaşımı diğer yaklaşımlara göre en avantajlı olanıdır. Yazılmış olan kodlar ve mapping işlemleri tamamen geliştiricinin kontrolü altındadır. Projenizi çalıştırdığınızda ilgili yapıyı tetiklemeniz halinde kodlarla oluşturduğunuz classlar ve kurallara bağlı olarak veritabanınızı oluşturacaktır.

ENTITY FRAMEWORK YANSIMA ALMAK

İlk aşamada Entity Framework yapısını tanımak ve sorguları kullanmak için Database First yaklaşımı üzerinden başlayacağız. Sorgularımızı tanıdıktan sonra Model ve Database First yaklaşımlarını kullanacağız. Projemize hali hazırda bulunan veritabanı yansımaları almak için bir Entity Data Model (.edmx) dosyası eklemeliyiz. EDM üç bölümden (Conceptual Model, Mapping, Storage Model) oluşmaktadır.

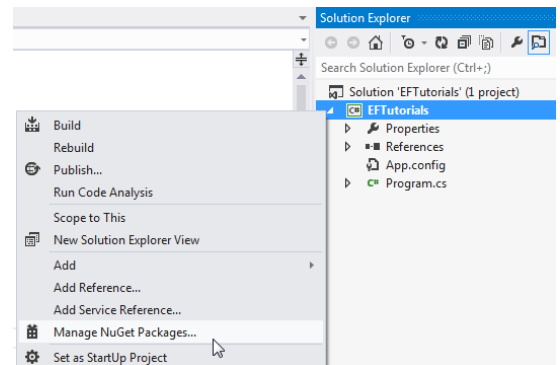
- **Conceptual Model:** Bu alanda model sınıflarımız ve bu sınıfların ilişkileri yer alacaktır. Bu sınıflar veritabanı tasarımınızdan bağımsız olacaktır.
- **Storage Model:** Bu alanda veritabanı tasarım modelimiz yer alır. Bu model içerisinde veritabanımıza ait tablolar, view'lar, stored procedure'ler ve bunlara ait ilişkiler ve key'ler yer alır.
- **Mapping:** Bu alan ise model sınıflarımız ile tasarım modelimiz arasındaki haritalama işlemlerinin bulunduğu alandır.

Veritabanının yansımaları almak, MsSQL'de bulunan veritabanımızdaki schema bilgisinin visual studio tarafına yansıtılmasıdır. Visual studio MsSQL'den gelen tabloları class olarak tanır, tablolarda bulunan kolonları ise property olarak okur. Bir proje içerisinde veritabanının yansımaları almak için aşağıdaki adımları takip edebilirsiniz.

- Solution Explorer penceresinden Projenizin içerisindeki References'a sağ tıklayın.
- Manage NuGet Package seçeneğini seçin. Açılan pencereden "Entity Framework" kütüphanesini bulun ve projenize yükleyin. References içerisinde Entity Framework kütüphanesine ait dosyalar gelecektir.

Yukarıdaki adımları takip ettiğimizde sağdaki pencere açılacaktır. NuGet Packages penceresi microsoft'un nuget.org sitesine bağlanarak internet üzerinden .Net Framework içerisine dahil olmayan bir kütüphaneyi projenize dahil etmek için kullanılır. Ayrıca .Net Framework içerisinde bulunan bir kütüphanenin daha güncel bir versiyonunda nuget üzerinden indirebilirsiniz.

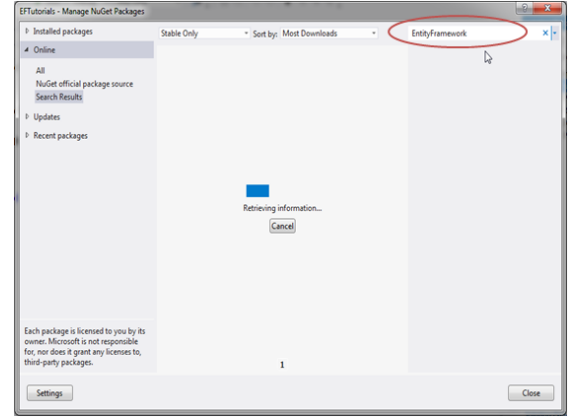
Entity Framework kütüphanesi .Net Framework versiyonuna göre içerisinde Entity Framework kütüphanesinin çeşitli versiyonlarını barındırır.



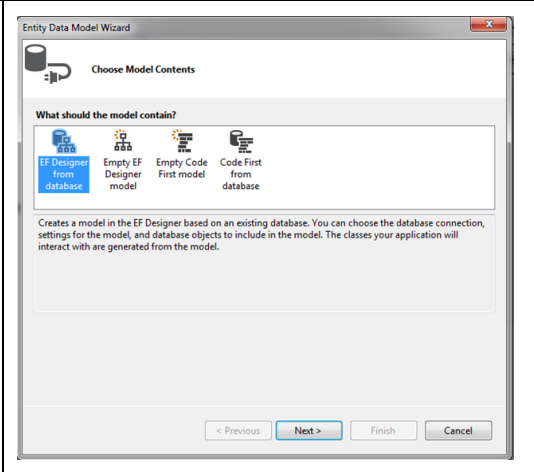
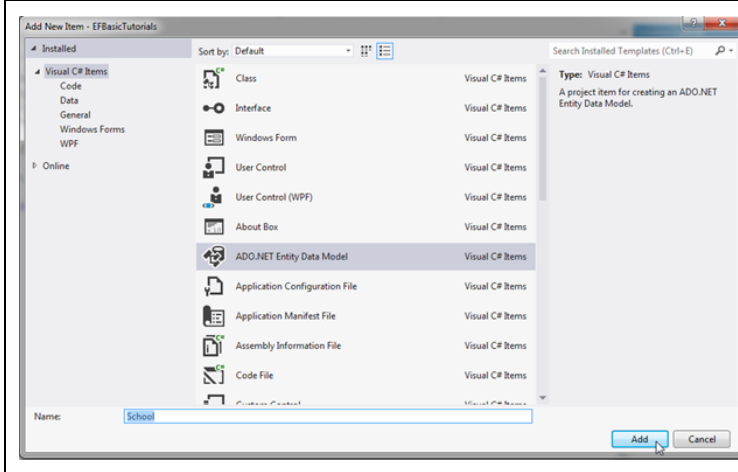
Entity Framework veya diğer kütüphaneleri kullanırken her zaman en güncel versiyonlarını kullanmanızı tavsiye ederiz.

Her yeni versiyonda kütüphane içerisinde bir çok güncelleme ve performans iyileştirmeleri yapılmaktadır. Bu güncellemeler ile güvenli ve performanslı uygulamalar geliştirebilirsiniz.

Bu işlemleri tamamladıktan sonra aşağıdaki adımlar ile devam edebilirsiniz.



- Solution Explorer penceresinden Proje Adınıza sağ tıklayın.
- Add > New Item diyerek yeni bir bileşen eklemek için pencereyi açıyoruz.
- Sol tarafta bulunan Data sekmesine geçiyoruz.
- ADO.NET Entity Data Model seçeneğini seçip Tamam butonuna tıklıyoruz.
- Bu işlemi tamamladıktan sonra “Entity Data Model Wizard” penceresi açılacaktır.

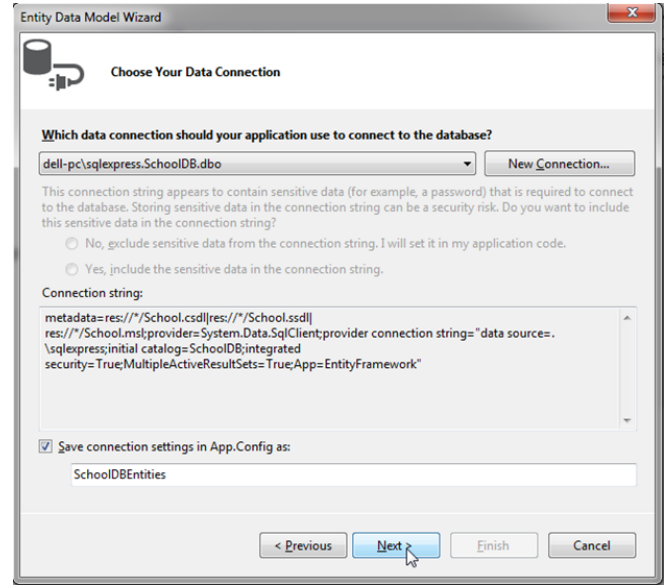


Wizard penceresinden “EF Designer from database” seçeneğini seçerek veritabanı yansımaları alabilirsiniz. Entity Framework yaklaşımlarından ilk olarak “Database First” yaklaşımı ile ilerleyeceğimizden dolayı “EF Designer from database” seçeneğini seçerek ilerliyoruz.

“EF Designers from database” seçeneğini seçip ileri butonuna tıkladığımızda “Choose Your Data Connection” penceresi ile karşılaşacağız. Bu pencere kullanacağımız veritabanına ulaşmak için gerekli bağlantı bilgilerini istemektedir. Verdiğiniz bilgilere göre Entity Framework bir connection string oluşturacaktır.

“New Connection” butonuna tıklayarak yeni bir bağlantı oluşturabilirsiniz. Açılan “New Connection” penceresinden bağlanmak istediğiniz sql host adresi (local host olarakda kullanabilirsiniz), sql hizmetine bağlantı türünüz, Sql Server Authentication seçeneğini seçerseniz sql hizmetine bağlanmak için gerekli kullanıcı adınız ve şifrenizi girerek bağlantı oluşturabilirsiniz.

Bağlantı bilgileriniz doğruysa kullanmak istediğiniz veritabanını combobox’dan seçerek tamam butonuna tıklayın.



The dialog box titled "Entity Data Model Wizard" has a tab "Choose Your Data Connection". It asks "Which data connection should your application use to connect to the database?". A dropdown menu shows "deli-pc/sqlexpress.SchoolDB.dbo". A "New Connection..." button is on the right. Below, a warning message states: "This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?". Two radio buttons are present: "No, exclude sensitive data from the connection string. I will set it in my application code." (selected) and "Yes, include the sensitive data in the connection string.". A text area labeled "Connection string:" contains a long string starting with "metadata=res://*/School.csdl|res://*/School.ssdl|res://*/School.mst;provider=System.Data.SqlClient;provider connection string='data source=...'. A checkbox "Save connection settings in App.Config as:" is checked, with "SchoolDBEntities" entered in the field below. At the bottom are buttons: "< Previous", "Next >" (highlighted), "Finish", and "Cancel".

Tamam butonuna tıkladığınızda “New Connection” penceresi kapanacaktır.

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?"

Bu bağlantı dizesi, veritabanına bağlanmak için gereken hassas verileri (örneğin, bir parola) içeriyor gibi görünüyor. Bağlantı dizesindeki dizgeye duyarlı veriler bir güvenlik riski oluşturabilir. Bu hassas verileri bağlantı dizesine eklemek istiyor musunuz? "

alanı aktifleştirecektir. Connection string içerisinde parola gibi verileri saklamak isteyip, istemediğimizi sormaktadır. İlk seçenek ve çevirisi aşağıdadır.

No, exclude sensitive data from the connection string. I will set it in my application code.

Hayır, hassas verileri bağlantı dizesinden hariç tutun. Bunu uygulama içerisinde kod olarak ayarlayacağım.

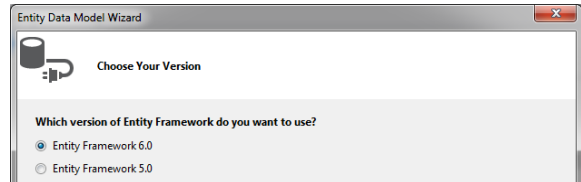
İkinci seçeneğimiz ve çevirisi ise aşağıdadır.

Yes, include the sensitive data in the connection string.

Evet, hassas verileri bağlantı dizesine ekleyin.

RadioButton’ları kullanarak seçimimizi yapıyoruz. Biz örneğimizde “Yes, include the sensitive data in the connection string” seçeneğini seçiyoruz. İleri butonuna tıklayarak wizard ekranındaki sıradaki pencereye geçiş yapabiliriz.

“Choose Your Version” penceresinde kullanılacak entity framework versiyonunu seçmemizi istemektedir. Eğer Nuget üzerinden Entity Framework yüklemeyerseniz buradan “Entity Framework 6.0” seçeneğini seçerek ilerleyebilirsiniz.



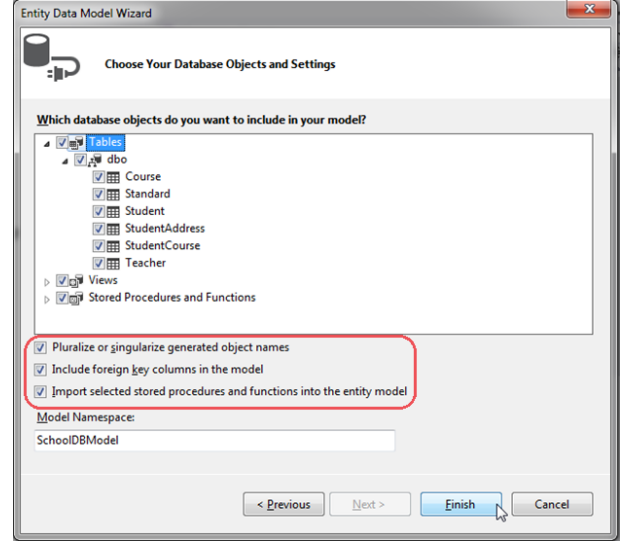
The dialog box titled "Entity Data Model Wizard" has a tab "Choose Your Version". It asks "Which version of Entity Framework do you want to use?". There are two radio buttons: "Entity Framework 6.0" (selected) and "Entity Framework 5.0".

Database first yaklaşımını kullandığımızda hali hazırda SQL hizmetimiz üzerinde bulunan bir veritabanını kullanacağımızdan daha önce bahsetmiştik. “Choose Your Database Objects and Settings” penceresinden hali hazırda bulunan veritabanımızdan kullanacağımız veritabanı nesnelerini seçerek yansıma alma işlemi sırasında C# tarafından kullanabileceğimiz bir formatta yazdırılması gerekmektedir.

Hemen aşağıda bulunan checkboxları kullanarak yansıma alma işlemi ile ilgili ayarları değiştirebiliriz.

Dikkat etmemiz gereken bir diğer önemli nokta ise Model Namespace alanıdır. Kullanacağımız modelin namespace’ini değiştirebilirsiniz. Örneklerimizde biz namespace bilgisini varsayılan olarak bırakacağız.

Diğer ayar seçeneklerini değiştirmeden “Pluralize or singularize generated object names” seçeneğini işaretleyin.



Veritabanı yansıması alınırken yapabileceğimiz ayarların açıklamalarını yaparak devam edelim.

Pluralize or singularize generated object names

Oluşturulan nesne adlarını çoğullaştırma ve tekilleştirme

Bu seçenek veritabanınızda çoğul olarak yazdığınız tablo isimlerini class olarak yazdırırken tekilleştirilmesini sağlar. Object Oriented’den bildiğiniz gibi class oluşturma standartlarına göre bir class ismi büyük harf ile başlamalı ve tekil olmalıdır. Bu seçeneği işaretlediğinizde tekilleştirme standartını uygulamaktadır. Önemli bir nokta olarak bu özelliğin çalışabilmesi için veritabanınızda ki tabloların ingilizce olarak isimlendirilmesi gerekmektedir.

Include foreign key columns in the model

Model de anahtar sütunları dahil et

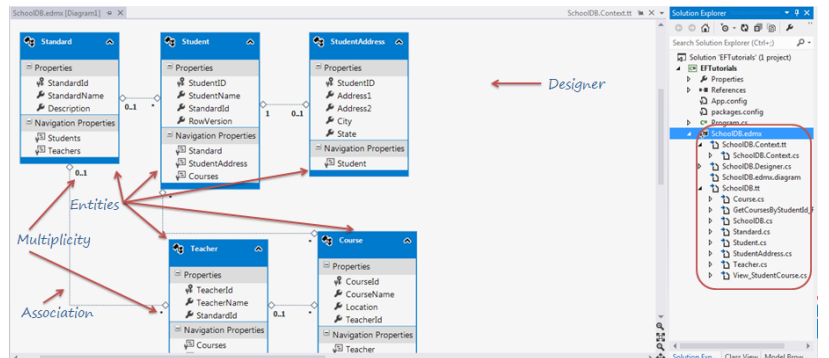
Bu seçeneği işaretlersek özellikler arasında foreign key olan özelliği kullanarak diğer alt özelliklere erişim imkanı sağlanmaktadır.

Import selected stored procedures and functions into the entity model.

Seçili saklı yordamları ve fonksiyonları varlık modeline aktarın.

Bu seçenek ile yansıma alınırken stored procedure ve function gibi veritabanı nesnelerinin yansımanın içerisine aktarılmasını sağlayabilirsiniz.

Son olarak pencerede bulunan “Finish” butonuna tıklayarak yansıma alma işlemi tamamlayabilirsiniz. Yansıma alma işlemi tamamlandıktan sonra veritabanı diagramını gösteren bir sayfa açılacaktır.



Solution Explorer içerisinde bulunan edmx dosyamız içerisinde veritabanına ait tüm yapılar bulunmaktadır. Ayrıca gerekli konfigürasyon ayarlarında app.config/web.config dosyaları içerisinde yapılacaktır. Örnek konfigürasyon ayarları dosyası aşağıda bulunmaktadır.

App.config

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false"/>
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"/>
  </startup>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework"/>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer"/>
    </providers>
  </entityFramework>
  <connectionStrings>
    <add name="SchoolDBEntities"
connectionString="metadata=res://*/SchoolDB.csd|res://*/SchoolDB.ssd|res://*/SchoolDB.msl;provider=System.Data.SqlClient;provider connection string="data source=.\\sqlexpress;initial catalog=SchoolDB;integrated
security=True;multipleactiveresultsets=True;application name=EntityFramework";"
providerName="System.Data.EntityClient"/>
  </connectionStrings>
</configuration>
```

Yansıma aldıktan sonra artık veritabanı üzerinde veri sorgulamaları yapabiliriz. Bu sorgulamaları **LINQ to Entities** ile yapmaktayız. **LINQ to Entities** Nesneleri sorgulamada kullanacağımız sorgulama dilidir. Bu sorgular bize model sınıflarını döndürecektir. Bu alanda LINQ'in yeteneklerini de kullanabiliriz. Linq to Entities haricince **Entity SQL** dili bulunmaktadır. Yine LINQ to Entities gibi sorgulama yapabileceğimiz bir sorgulama dilidir. Ancak Linq To Entities'e göre biraz daha zordur.

ENTITY FRAMEWORK SORGULARI

Yansıma alma işlemimizi tamamladıktan sonra artık sorgularımızı yazmaya başlamadan önce bir adımımız kaldı. Veritabanımızın yansımasının instace'ını almak. Yansıma üzerindeki nesneleri ve yapıları kullanabilmemiz için kullanacağımız class içerisinde instance almamız gerekmektedir. Biz örneklerimizde "Northwind" veritabanını kullanacağız. Aşağıdaki işlemler "Northwind" veritabanının yansımasını aldığınız düşünülerek yazılmıştır.

Entity Framework'de Sorgu Yazmadan Önce Bilinmesi Gerekenler

Sorgu yazmaya geçmeden önce sorgularımızı oluşturmak için bilmemiz gereken iki kavram bulunmaktadır. Bu kavramlardan birincisi Extension metot, bir diğeri ise Lambda Expression'dır.

Extension Metot

Extension methods sadece static classlar içinde tanımlanabilen, parametre tanımlamalarında türden önce this ifadesini kullandığımız static metotlardır. Parametre başına yazılan this ifadesi metodun çağırılma yöntemini zincirleme metot olarak değiştirecektir. Extension metot ile ilgili bir örnek aşağıda yer almaktadır.

Örnek Bir Extension Metot

```
//Metodun static olarak işaretlenmesine dikkat edin.  
public static int ToInt32Ext(this string s) //this keyword'ü ile extension metot tanımlıyoruz.  
{  
    return Int32.Parse(s);  
}
```

Kullanımı

```
string s = "9";  
  
int i = s.ToInt32Ext(); //Extension method kullanımı  
label1.Text = i.ToString();
```

Kullanırken metot parantezleri arasında parametre göndermek yerine, tipimizin sonuna .ToString metodunu ekler gibi, oluşturduğumuz metodu ekledik.

Lambda Expression

Lambda İfadeleri, değişkenlere değer atamak için kullanılan sadeleştirilmiş isimsiz metotlardır. Bu metotlar matematik'de ki Lambda Calculus'un C# kullanımıdır.

Lambda İfadeleri kullanarak parametre geçilebilen ve değer döndüren isimsiz yerel metotlar oluşturabilirsiniz. Bu ifadeler genelde basit işlemleri bildirmekte kullanılabilir ve LINQ sorgularının yazımında çok işe yararlar.

Lambda Expression

```
n => n * n //Okunuşu : n öyleki n * n
```

Yansımanın Instance'ını Almak

Veritabanı Yansımasından Instance Almak

```
NorthwindEntities db = new NorthwindEntities();
```

Entity Framework yansıması üzerinden işlemlerimizi yürütebilmemiz için öncelikle yapıları RAM üzerinde tanımlamamız gerekmektedir. Bu işlemden sonra artık veritabanı üzerinde CRUD işlemleri ve detaylı sorgulama işlemleri yapabiliriz.

Veritabanından Veri Çekmek ve Listelemek

Formunuza bir adet data grid view kontrolü ekleyin ve Form_Load eventine aşağıdaki kodları yazın. Aşağıdaki kod veritabanınızda seçtiğiniz tablonun tüm satırlarını listelemenizi sağlayacaktır.

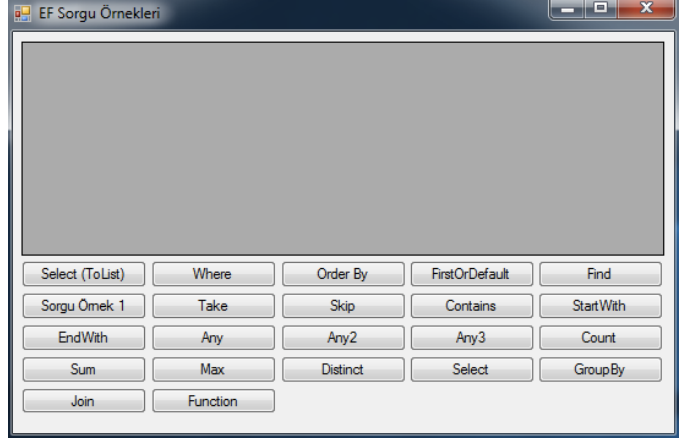
Veritabanında Bulunan Tablodan Verileri Çekmek

```
dataGridView1.DataSource = db.Categories.ToList();
```

Entity Framework ile Select İşlemleri

Entity Framework kütüphanesini kullanarak veritabanımızda bulunan bir tablonun üzerinde sorgulamalar yaparak verileri filtreleyebiliriz. Hemen hemen T-SQL üzerinde yapabildiğiniz tüm sorgular Entity Framework ile yapılabilmektedir. Sırası ile örnekler üzerinden bir kaç sorgu örneği yapalım. Tüm örneklerimiz “Northwind” veritabanı üzerinden yapılacaktır. Örneklerimiz için yandaki form görünümünü hazırlayın.

Resimde ki Form Tasarımı temsildir, örnek kodlamaları ve farklı sorgular yapmak için daha fazla button kullanabilir veya farklı kontrolleri tercih edebilirsiniz.



EF Sorgu Örnekleri formu, sorgu oluşturma için kullanılan bir araçtır. Formun üst kısmında 'Select (ToList)', 'Where', 'Order By', 'FirstOrDefault', 'Find' gibi butonlar yer almaktadır. Aşağıda 'Sorgu Örnek 1' başlığı altında 'Take', 'Skip', 'Contains', 'StartWith', 'EndWith', 'Any', 'Any2', 'Any3', 'Count', 'Sum', 'Max', 'Distinct', 'Select', 'GroupBy', 'Join', 'Function' gibi butonlar yer almaktadır.

ToList() Metodu

ToList() metodu veritabanında bulunan tabloda ki tüm ürünleri tablonun yansımada ki nesnesinin tipinde bir generic list hazırlaması için kullanılır. Aşağıdaki sorgunun sonucunda List<Supplier> tipinde dönüş yapacaktır. ToList() metodu yazılmadığında sorgu hata verecektir. ToList() bir extension metod olduğundan tablo sonuna . yazıldıktan sonra eklenebilir, parametre parantezlerinin boş olması ToList metodunun parametre almadığını gösterir.

Örnek 1 : Northwind veritabanında ki tüm tedarikçileri listeleyin.

```
dgvListe.DataSource = db.Suppliers.ToList();
```

Where Metodu

Veritabanınızda bulunan tablolarda filtreleme işlemi yapmak için kullanılır. Where extension metodunu kullanarak filtreleme işlemi yapıyoruz. X ifadesi lambda expression yönteminde Products'ı temsil etmektedir. Where metodu içerisinde Products tablosunu kullanacağımız zaman bu tabloyu x ifadesi ile çağıracağımızı belirtiyoruz. x öyle ki tanımlamasını yaptıktan sonra x.Property seçimi yaparak ilişkisel sorgular yapabiliriz. Where metodu içerisinde birden fazla ilişkisel sorguyu, mantıksal operatörlerle birleştirerek kullanabiliriz.

Örnek 2 : Northwind veritabanında fiyatı 20'den fazla olan ürünlerin bilgilerini getirin.

```
dgvListe.DataSource = db.Products.Where(x => x.UnitPrice > 20).ToList();
```

İki ilişkisel sorguyu mantıksal operatörleri kullanarak birleştirerek aşağıdaki gibi bir sorgu elde edebiliriz.

Örnek 3 : Northwind veritabanında fiyatı 20 ile 50 arasında olan ürünlerin (Products tablosu) ProductID, ProductName, UnitPrice, UnitsInStock bilgilerini getirin.

```
dgvListe.DataSource = db.Products.Where(x => x.UnitPrice > 20 && x.UnitPrice < 50).ToList();
```

OrderBy – OrderByDescending Metotları

T-Sql'de bulunan Order By deyimi ile aynı görevi üstlenmektedir. Sorgu sonucuna göre gelen generic list içerisinde sıralama işlemleri yapmaktadır. İki farklı metod olarak kullanılmaktadır. Order By metodu artarak sıralama işlemini üstlenmekte, OrderByDescending ise azalarak sıralama işlemini üstlenmektedir. Örnek kullanımları aşağıdadır.

Örnek 4 : Northwind veritabanındaki ürünleri ürün ismine göre alfabetik olarak sıralayın.

```
dgvListe.DataSource = db.Products.OrderBy(p => p.ProductName).ToList();
```

Örnek 5 : Ürünleri fiyatına göre tersten sıralayın. (En yüksek fiyat ilk sıraya gelecek şekilde..)

```
dgvListe.DataSource = db.Products.OrderByDescending(p => p.UnitPrice).ToList();
```

First Metodu

First komutu bir koleksiyon içerisinde ki ilk elemanı seçmek için kullanılmaktadır. Eğer bir kriter oluşturursak ve kriter sonucunda bir koleksiyon dönüyorsa verilen kritere göre oluşturulan koleksiyonda ki ilk eleman döndürülür.

Yandaki resimde Northwind veritabanında ki Categories tablosuna bir select işlemi uygulandı. Tablomuzda sekiz veri satırı bulunmaktadır. First komutunu kullanarak sonuçları inceleyelim.

Aşağıdaki sorgu sonucunda tüm kategorileri listelediğimiz için dönen generic list'de bulunan ilk kayıt (1 ID'li Beverages) ekranda gösterilecektir.

Categories Tablomuz

CategoryID	CategoryName
1	Beverages
2	Condiments
3	Confections
4	Dairy Products
5	Grains/Cereals
6	Meat/Poultry
7	Produce
8	Seafood

First metodunu daha iyi kullanabilmek için üzerinde çeşitli örnekler yapalım.

Örnek 6 : İlk kaydı elde etmek

```
Category cat = db.Categories.First();  
if (cat == null)  
    MessageBox.Show("Böyle bir kategori bulunamadı!");  
else  
    MessageBox.Show(cat.CategoryName); //Örneğimize Göre Ekranda Beverages Görünecektir.
```

Verilen bir kritere göre ilk kaydı elde etmek istediğimizde yine First metodunu kullanabiliriz.

Örnek 7 : Verilen şarta göre ilk kaydı elde etmek

```
Category cat = db.Categories.First(x => x.CategoryID > 6);  
if (cat == null)  
    MessageBox.Show("Böyle bir kategori bulunamadı!");  
else  
    MessageBox.Show(cat.CategoryName); //Örneğimize Göre Ekranda Produce Görünecektir.
```

First metodu kullanılırken dikkat edilmesi gereken en önemli nokta ise First metodunu kullandığımızda dönen koleksiyon yapısı içerisinde en az bir veri olması gerekmektedir. Dönen değer içerisinde bir kayıt yoksa first "InvalidOperationException" hatası verecektir.

Örnek 8 : Verilen şarta uygun kayıt olmadığında hata verir.

```
Category cat = db.Categories.First(x => x.CategoryID > 20); //Bu sorguyu çalıştırdığımızda First  
sətırında Runtime hatası verecektir ve aşağıdaki kodlar çalıştırılmayacaktır. Bunun sebebi  
kategoriler tablosundaki en son id değeri 8'dir. Sorgumuzda ise 20'den büyük olanları listele ve  
bunlardan ilk sırada olanı getir şeklinde bir sorgu tanımladık. Verdiğimiz şarta uygun veri  
olmadığından hata verecektir.  
if (cat == null)  
    MessageBox.Show("Böyle bir kategori bulunamadı!");  
else  
    MessageBox.Show(cat.CategoryName);
```

FirstOrDefault Metodu

FirstOrDefault, First ile aynı kullanım amacına sahiptir. Farklı olarak FirstOrDefault içerisinde sağlanmayan bir şart verildiğinde First metodunda ki gibi hata vermeyecektir. Sadece veritabanında eşleşen bir kayıt bulunamadığından Category nesnesini null olarak bırakacaktır. First'de aşağıdaki if komut bloğu çalıştırılmıyorken FirstOrDefault komutunda if kontrolü içerisinde nesnenin null olup olmadığı sorgulanabilmektedir.

Örnek 9 : FirstOrDefault Kullanımı

```
Category cat = db.Categories.FirstOrDefault(x=> x.CategoryID > 7);
if (cat == null)
    MessageBox.Show("Böyle bir kategori bulunamadı!");
else
    MessageBox.Show(cat.CategoryName); //Ekranda Seafood yazısı görünecektir.
```

First metodunda yaptığımız gibi veritabanından dönen koleksiyonda olmayan bir şart girdiğimizde FirstOrDefault metodunun nasıl davrandığını incelemek için aşağıdaki komutu yazalım.

Örnek 10 : FirstOrDefault Kullanımı

```
Category cat = db.Categories.FirstOrDefault(x=> x.CategoryID > 20);
if (cat == null)
    MessageBox.Show("Böyle bir kategori bulunamadı!"); //Nesne bulunamadığından null olarak
dönecektir. Karar yapımız sayesinde ekranda messagebox göstereceğiz.
else
    MessageBox.Show(cat.CategoryName);
```

First ve FirstOrDefault arasında ki temel fark verilen hatalardan kaynaklanmaktadır.

Find Metodu

Microsoft SQL Server üzerinde veritabanı işlemlerini incelerken Identity Key (ID) değerinden söz etmiştik.. Kısaca hatırlamak gerekirse ID değeri her bir veri satırının kimlik kolonu olarak görev almaktadır. Uniq olduğundan tek bir kayıt'a erişmek için ID kolonu kullanılmaktaydı. Entity Framework kütüphanesini kullanarak veritabanında bulunan tablolarınızda ki tek bir kolona erişmek için aşağıdaki komutu kullanabilirsiniz.

Find Metodu

```
Category gelenKategori = db.Categories.Find(<id>); //id parametresi veritabanından getirmek
istediğiniz kayıdın id'si olmalıdır.
```

Eğer veritabanında verdiğiniz ID'ye ait kayıt yoksa gelenKategori isimli nesne null değerine sahip olacaktır. Find metodunu aşağıdaki örnekte olduğu gibi kolayca kullanabilirsiniz.

Örnek 11 : ID'si 1 olan ürünün bilgilerini getirin.

```
//Primary kolonunda arama yapmak için Find metodu kullanılır.
Product yakalanan = db.Products.Find(1);
MessageBox.Show(yakalanan.ProductName);
```

Şu ana kadar gördüğümüz Entity Framework sorgu metotlarını kullanarak aşağıdaki örneği yapmaya çalışalım.

Örnek 12 : Kategori ve Tedarikçi ID'si 1 olan ürünleri isme göre tersten sıralayın.

```
dgvListe.DataSource = db.Products.Where(p => p.CategoryID == 1 && p.SupplierID ==
1).OrderByDescending(p => p.ProductName).ToList();
```

Take Metodu

T-SQL sorgularında kullandığımız Top komutunun görevini üstlenmektedir. Bir sorgu sonucunda dönen listeden en üstteki belirlediğimiz sayıda satırı ekranda göstermemizi sağlar.

Örnek 13 : En pahalı 5 ürünü getirin.

```
dgvListe.DataSource = db.Products.OrderByDescending(p => p.UnitPrice).Take(5).ToList();
```

Skip Metodu

Dönen sorgu sonucundan en üstteki x kadar satırı atlayarak o satırlardan sonraki belirlediğimiz sayıda satırı ekranda göstermemizi sağlayan metottur. Aşağıdaki örnekte öncelikle UnitPrice kolonuna göre tersten sıralama işlemi yapıyoruz. Dönen listeden Skip Komutu ile 5 tanesini atlıyoruz ve son olarak Take 5 komutu ile kalan listedeki en üstten 5 kaydı ekranda gösteriyoruz.

Örnek 14 : En pahalı ilk 5 üründen sonraki en pahalı 5 ürünü getir.

```
dgvListe.DataSource = db.Products.OrderByDescending(p => p.UnitPrice).Skip(5).Take(5).ToList();
```

ID	Name	UnitPrice
38	Côte de Blaye	263.5000
80	qqqqq	222.0000
29	Thüringer Rostbratwurst	123.7900
9	Mishi Kobe Niku	97.0000
20	Sir Rodney's Marmalade	81.0000
18	Carnarvon Tigers	62.5000
59	Raclette Courdavault	55.0000
51	Manjimup Dried Apples	53.0000
62	Tarte au sucre	49.3000
43	Ipoh Coffee	46.0000

Yukarıdaki sorgu sonucunda ilk 5 ürün (ID'si 18 olan ürüne kadar) atlanacak 18, 59, 51, 62 ve 43 ID'li ürünler ekranda gösterilecektir.

Contains Metodu

T-SQL örneklerinde kullandığımız LIKE komutu mantığında çalışır. Like sorgularından biraz daha temel ve basit sorgular yapmak için tasarlanmıştır. Verilen ifadenin belirtilen kolon içerisinde geçip geçmediğine bakar.

Örnek 15 : Ürün adı içerisinde 'a' harfi geçen ürünleri getirin.

```
dgvListe.DataSource = db.Products.Where(p => p.ProductName.Contains("a")).ToList();
```

StartsWith ve EndsWith Metotları

Like sorgularındaki başlayan ve biten karakterlere göre aramalar yapmamızı sağlayan metotlardır. Belirtilen kolonun, verilen ifade ile başlayıp başlamadığına, bitip bitmediğine göre sonuç döndürür.

StartsWith başlangıç karakterini kontrol eder.

Örnek 16 : Adı c harfi ile başlayan ürünleri listeleyin.

```
dgvListe.DataSource = db.Products.Where(p => p.ProductName.StartsWith("c")).ToList();
```

EndsWith ise bitiş karakterini kontrol eder.

Örnek 17 : Adı a harfi ile biten ürünleri listeleyin.

```
dgvListe.DataSource = db.Products.Where(p => p.ProductName.EndsWith("a")).ToList();
```

Any Metodu

İki farklı kullanıma sahip olan Any metodu ilk kullanımı olan bir tabloda kayıt olup olmadığını kontrol edebilir. İkinci kullanımı ise tabloda verilen şartlar uygun kayıt olup olmadığını kontrol etmektedir. Any geriye sonuç olarak boolean veri tipinde dönüş yapmaktadır.

Örnek 18 : Kategoriler tablosunda herhangi bir kayıt varmı?

```
bool sonuc = db.Categories.Any();  
MessageBox.Show(sonuc.ToString());
```

Any metodunun ikinci kullanışı ise tabloda verilen şarta uygun bir kayıt olup olmadığını öğrenmektir.

Örnek 19 : Kategoriler tablosunda “Beverages” isimli bir kategori kayıtlımı?

```
bool sonuc = db.Categories.Any(c => c.CategoryName == "Beverages");  
MessageBox.Show(sonuc.ToString());
```

Any metodunu farklı metotlarla birliktede kullanabilirsiniz.

Örnek 20 : Kategoriler tablosunda adı “Be” ile başlayan bir kategori varmı?

```
bool sonuc = db.Categories.Any(c => c.CategoryName.StartsWith("Be"));  
MessageBox.Show(sonuc.ToString());
```

Entity Framework ile Aggregate Function Kullanımı

T-SQL sorgularının ve raporlamanın vazgeçilmezi olan bütünleşik fonksiyonları Entity Framework kütüphanesi ilede kullanabilirsiniz.

Count Metodu

T-SQL’den de hatırlayabileceğiniz gibi Count metodu sorgu sonucunda dönen result’un satırlarını saymaktadır. Count metodu geriye int değer tipinde dönüş yapmaktadır.

Örnek 21 : Ürünler tablosunda kaç ürünüm bulunmaktadır?

```
int urunSayisi = db.Products.Count();  
MessageBox.Show(urunSayisi.ToString());
```

Sum Metodu

T-SQL’de kullanılan SUM metodu ile aynı görevi üstlenir. Dönen result içerisinde belirlediğiniz kolonların değerlerinin toplanmasını sağlar.

Örnek 22 : Tüm ürünler dahil olmak üzere stoğumda kaç adet ürün var? (Her ürünün stok değerlerini toplamak istiyoruz.)

```
decimal? fiyat = db.Products.Sum(p=> p.UnitsInStock);  
MessageBox.Show(fiyat.ToString());
```

Max - Min Metotları

Bir kolon içerisindeki maximum ve minimum değerleri bulmak için kullanacağınız metotlardır.

Örnek 23 : En pahalı ürün fiyatı nedir?

```
decimal? enYuksekFiyat = db.Products.Max(p => p.UnitPrice);  
MessageBox.Show(enYuksekFiyat.ToString());
```

Örnek 24 : En ucuz ürün fiyatı nedir?

```
decimal? enDusukFiyat = db.Products.Min(p => p.UnitPrice);  
MessageBox.Show(enDusukFiyat.ToString());
```

Distinct Metodu

T-SQL'de ki gibi tekrar eden değerleri egale etmek için kullanılır.

Örnek 25 : Hangi ülkelerle iş yapıyoruz? Ülke listesini çıkartın.

```
List<string> liste = db.Orders.Select(o => o.ShipCountry).Distinct().ToList();  
foreach (string item in liste)  
{  
    listBox1.Items.Add(item);  
}
```

Select Metodu

Select metodu sorgularınızı çekerken istediğiniz kolonları almanıza veya kolon isimlerinizi kullanıcılara göstermemek için değiştirmenize olanak sağlar.

Örnek 26 : Müşterilerin müşteri adı, yetkilisi, telefonu ve adresini farklı kolon isimleriyle getiren sorguyu yazın.

```
dgvListe.DataSource = db.Customers.Select(c => new {  
    MusteriSirketi = c.CompanyName,  
    Yetkili = c.ContactName,  
    Telefon = c.Phone,  
    Adres = c.Address  
}).ToList();
```

Group By Metodu

Sorgularımızdan dönen sonuçları belli kriterlere göre gruplamamızı sağlar.

Örnek 27 : Hangi kategoride kaç adet ürünüm var?

```
dgvListe.DataSource = db.Products.GroupBy(p => p.Category.CategoryName).Select(g => new {  
    KategoriAdi = g.Key,  
    ToplamStok = g.Sum(p => p.UnitsInStock)  
}).ToList();
```

Join İşlemleri

T-SQL sorguları yazarken select sorgularımızda birden fazla tablodan veri çekmemiz gereken durumlarla karşılaşabiliyorduk. Bu işlemler için uzun ve karışık join sorguları yazmamız gerekmektedir. Entity Framework kütüphanesini kullanırken bir verinin başka bir tablo ile olan bağlantısı üzerinden (join yaparak) veri çekmek oldukça kolay bir işlemdir.

Örnek 28 : Ürünlerin ürün adını, fiyatını, stok miktarını, kategori adını ve tedarikçi adını getiren sorguyu yazınız.

```
dgvListe.DataSource = db.Products.Select(p => new {
    UrunAdi = p.ProductName,
    Fiyat = p.UnitPrice,
    Stok = p.UnitsInStock,
    Kategori = p.Category.CategoryName,
    Tedarikci = p.Supplier.CompanyName
}).ToList();
```

Function Kullanımı

Entity Framework kütüphanesini kullanırken T-SQL'de bulunan belli başlı fonksiyonları kullanmanız gerektiği durumlarda bu fonksiyonları kolayca kullanabilirsiniz.

Örnek 29 : Çalışanların yaşlarını hesaplayın.

```
dgvListe.DataSource = db.Employees.Select(x => new {
    Adi = x.FirstName,
    Soyadi = x.LastName,
    DogumTarihi = x.BirthDate,
    Yasi = SqlFunctions.DateDiff("Year", x.BirthDate, DateTime.Now)
}).ToList();
```

Entity Framework kütüphanesi içerisinde bu örneklerimiz dışında bir çok metot ve özellik bulunmaktadır. Tüm metot kullanımları benzer özelliklere sahiptir.

Veritabanına Insert İşlemi Yapmak

Northwind veritabanında ki kategoriler tablosuna yeni bir kayıt ekleyelim. Bu işlem için instance aldığınızı varsayıyoruz. Ekleme işlemi yapmadan önce veritabanına ekleme yapacağımız tablo için bir nesne oluşturuyoruz (örneğimizde bir kategori nesnesi) ve bilgilerini dolduruyoruz.

Category sınıfından bir nesne oluşturuyoruz.

```
Category yeniKategori = new Category();
yeniKategori.CategoryName = txtKategoriAdi.Text;
yeniKategori.Description = txtAciklama.Text;
```

Category tipinde ki nesnemiz oluştuğuna göre artık nesnemizi veritabanı üzerinde bulunan Categories tablosuna ekleyebiliriz. Bu işlem için;

Yansıma'da ki Categories tablosuna yeniKategori nesnesini ekleyin.

```
db.Categories.Add(yeniKategori);
```

Yukarıdaki kodu derleyip çalıştırdığınızda hiç bir hata ile karşılaşmayacaksınız. Fakat veritabanınızdaki tabloyu kontrol ettiğinizde verinin eklenmediğini göreceksiniz. Verinin eklenmeme sebebi ekleme işleminin yansıma üzerinde yapılmış olmasından kaynaklıdır.

.Add() : Add metodu Entity Framework kütüphanesi içerisinde bulunan bir extension metotdur. Parametre olarak .Add metodundan önce yazılan tablo adına uygun veri tipinde bir varlık parametresi ister. Categories.Add yazıldığında Category tipinde parametre beklemektedir. Products tablosu ile işlem yapıldığında Product tipinde bir nesneyi parametre olarak isteyecektir.

Yaptığınız değişiklikleri yansıma üzerinden SQL Servisiniz üzerinde bulunan veritabanınıza aktarmak için yazmanız gereken son bir komut kaldı.

Yansımada ki Değişiklikleri Kaydet

```
db.SaveChanges();
```

Yukarıdaki komutuda yazdıktan sonra başarılı bir şekilde veritabanına eklendiğini göreceksiniz. Yukarıdaki ekleme işlemini daha az kod yazarak yapmamızda mümkündür. Aşağıda farklı iki yöntem bulabilirsiniz.

2. Ekleme Yöntemi (Farklı kullanım örnekleri için farklı tablo ve classlar kullanılmıştır.)

```
//2.Yöntem
Contact yeniKisi = new Contact { Adi = "Alp", Soyadi = "Kurtboğan" }; //object initialize
db.Contacts.Add(yeniKisi);
db.SaveChanges();//Değişiklikleri Kaydet
```

3. Ekleme Yöntemi

```
db.Contacts.Add(new Contact { Adi = "Alp", Soyadi = "Kurtboğan" });
db.SaveChanges();//Değişiklikleri Kaydet
```

Object initialize yöntemini kullanarak işlemlerinizi daha az satır kod yazarak yapabilmektesiniz. İlk kullanım ile ikinci kullanım arasında programın çalışmasında bir performans farkı olmayacaktır.

SaveChanges() Metodu

Değişiklikleri veritabanı üzerinde kaydetmek için kullanılan metottur. Bir ekleme, güncelleme veya silme işlemi yaptığınızda işleminiz öncelikle yansımanız üzerinde yapılacaktır. SaveChanges komutu yapılan değişikliklerin SQL hizmetinizdeki veritabanınıza üzerine kayıt edilmesini sağlar. Doğru kullanıldığında SaveChanges metodu performans artışı kazandıracaktır.

SaveChanged komutu olmaması durumunda birden fazla satır ekleme işlemi yapılacağını varsayarak;

- Veritabanı bağlantısı oluşturulur ve bağlantı sağlanır.
- Ekleme işlemi yapılır.
- Veritabanı bağlantısı kapatılır ve RAM üzerinden kaldırılır.

Bu işlem her eklenen kayıt için tekrar edecektir. Fakat ekleme işlemini yansıma üzerinde yaparak db.SaveChanges() komutu girildiğinde tüm satırlar (yansıma üzerinde yapılan değişiklikler) bir seferde veritabanı üzerine aktarılacaktır.

Veritabanında ki Bir Kaydı Güncellemek

Veritabanımızda ki bir kaydı güncellemek istediğimizde Find metodundan yardım alarak güncelleme işlemlerimizi yapabiliriz. Aşağıdaki örnek kod Categories tablosunda ki bir kaydı güncellemenizi sağlayacaktır.

Veritabanında Güncelleme İşlemi

```
//Veritabanından 1 ID'li kategoriye yakalıyoruz.
Category guncellenecek = db.Categories.Find(1);
//Nesnesnin yeni değerlerini atıyoruz.
guncellenecek.CategoryName = "Yeni Adı";
//Değişiklikleri kayıt ediyoruz.
db.SaveChanges();
```

Entry Metodu

Yukarıdaki yöntem ile bir kaydı kolayca güncelleyebilirsiniz. Fakat yukarıdaki güncelleme yöntemi performans kaybettiren bir yöntemdir. Nesne içerisinde bulunan propertylerden kaç tanesi değişirse değişsin nesneye propertylerine bağlı tüm kolonlar, kolonlarda değişiklik olmasa bile veritabanında güncellenecektir.

Bu performans sorununu aşmak için Entry metodunu kullanabiliriz. Entry metodu eski nesne ile yeni nesneyi karşılaştırarak sadece değişen kolonların update olmasını sağlamaktadır.

Entry metodu ile güncelleme yapmak

```
Category guncellenecek = db.Categories.Find(1);
db.Entry(db.Categories.Find(guncellenecek.CategoryID)).CurrentValues.SetValues(guncellenecek);
db.SaveChanges();
```

Veritabanındaki Kayıtları Silmek

Veritabanımızdan bir kayıt kaldırmak istediğimizde Entity Framework kütüphanesi içerisinde ki silme görevini üstlenen metotları kullanabiliriz.

Remove Metodu

Veritabanındaki bir kaydı silmek için Remove metodunu kullanabilirsiniz.

1 ID'li kategoriye silin.

```
//Silinecek kategori bilgilerini elde ediyoruz.
Category silinecek = db.Categories.Find(1);
//Yansıma üzerinde silme işlemini gerçekleştiriyoruz.
db.Categories.Remove(silinecek);
//Değişiklikleri kayıt ediyoruz.
db.SaveChanges();
```

Yukarıdaki sorguyu Northwind veritabanı üzerinde çalıştırdığınızda bir hata ile karşılaşacaksınız. Bu hata Categories tablosunda bulunan 1 ID'li kategoriye farklı tablolarla veya kolonlarla ilişkili veriler olduğundan silinemediği ile ilgili bir hatadır. Kategoriye silebilmek için öncelikle bu kategoriye bağlı olan kayıtları silmeniz gerekmektedir.

RemoveRange Metodu

Aynı kolon değerine sahip birden fazla kaydı tek bir komut satırı ile silmenizi sağlar.

OrderID'si 10248 olan siparişin detaylarını silin.

```
db.Orders.RemoveRange(db.Orders.Where(o => OrderID == 10248));
```

Entity Framework Sample

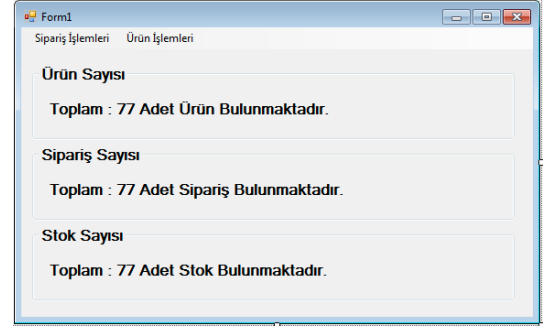
Temel Entity Framework sorgularını tamamladığımıza göre artık kapsamlı bir örnek yaparak öğrendiklerimizi kullanabiliriz. Örneğimiz için Windows Forms Application projesi kullanacağız.

Bu örneğin amacı, Öğrendiğimiz EF komutlarını kullanarak bir proje hazırlamak ve komutların çalışma mantığını anlamaktır. Bu örnekde sizden istenen Northwind veritabanı üzerinden ürün ve siparişlerin yönetiminin sağlanmasıdır. Örneğimizde dört Form ekranı bulunmaktadır.

Form1 Ekranı : Genel Bilgilendirme

Projenin açılış ekranıdır. Proje başlatıldığında veritabanımız ile ilgili bazı bilgiler sunacaktır. Bu bilgiler toplam ürün sayısı, toplam sipariş sayısı, toplam kategori sayısı, firmanın toplam cirosu, firmanın günlük cirosu gibi sorgu sonuçları olabilir. Firma hakkında bilgi verecek bir ekrandır. Form1 penceresinde MenuStrip kontrolü bulunacaktır. MenuStrip kontrolü ile diğer formlara geçiş sağlanacaktır.

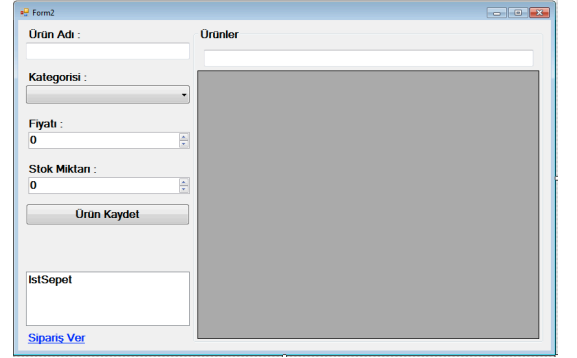
Form1 tasarımını yaparak projemize başlayabiliriz. Projemiz toplamda dört formdan oluşacaktır.

Form1 ekranı, 'Sipariş İşlemleri' ve 'Ürün İşlemleri' başlıklarına sahiptir. Ürün Sayısı: Toplam : 77 Adet Ürün Bulunmaktadır. Sipariş Sayısı: Toplam : 77 Adet Sipariş Bulunmaktadır. Stok Sayısı: Toplam : 77 Adet Stok Bulunmaktadır.

Form2 Ekranı : Ürün Yönetimi ve Sepet İşlemleri

Ürün ekleme işleminin ve ürün listeleme işleminin yapılacağı ekrandır. Form açıldığında tüm ürünler datagridview kontrolü içerisine listelenecektir. Sol panelden yeni ürün ekleme işlemi yapılacaktır. Aynı zamanda datagridview üzerinde bulunan textbox kontrolünden ürün arama işlemi yapılacaktır.

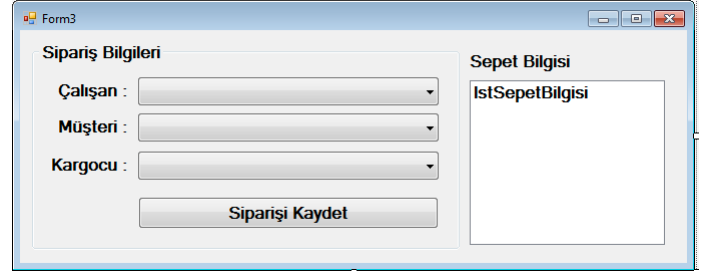
DataGridView kontrolü üzerinden çift tıklanan ürünü sol alt köşede bulunan Sepet Listbox kontrolüne ekleyeceğiz. Hemen altında bulunan sipariş ver butonu ile sepetde bulunan ürünlerin siparişi verilecektir.

Form2 ekranı, 'Ürün Adı', 'Kategori', 'Fiyat', 'Stok Miktar' alanları ve 'Ürün Kaydet' butonu içerir. Sağ tarafta 'Ürünler' datagridview kontrolü ve sol alt köşede 'IstSepet' listbox kontrolü yer almaktadır. Altta 'Sipariş Ver' butonu bulunur.

Form3 Ekranı : Sipariş İşlemleri

Ekranda bulunan formda sipariş bilgilerini seçerek bir sipariş oluşturacağız. Daha sonra Form2'de doldurduğumuz sepeti oluşturduğumuz siparişin detayları olarak veritabanına ekleyeceğiz.

Sipariş başarıyla kayıt edilirse Form4 açılacak ve Form4 içerisindeki DataGridView kontrollerinde siparişler ve sipariş detayları görüntülenecektir.

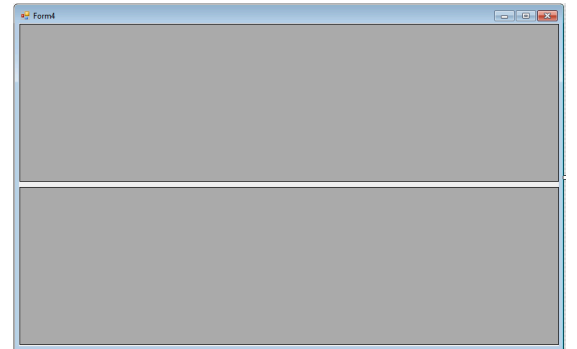
Form3 ekranı, 'Sipariş Bilgileri' ve 'Sepet Bilgisi' bölümlerine sahiptir. 'Sipariş Bilgileri' bölümünde 'Çalışan', 'Müşteri' ve 'Kargocu' alanları ve 'Siparişi Kaydet' butonu yer almaktadır. 'Sepet Bilgisi' bölümünde 'IstSepetBilgisi' listbox kontrolü bulunur.

Form4 Ekranı : Siparişleri Görüntüle

Form4 ekranında bulunan iki adet datagridview kontrolünden bir tanesi tüm siparişleri listeleyecektir. Bir siparişin üzerine çift tıkladığımızda ikinci datagridview kontrolünde çift tıklanan siparişe ait sipariş detayları listelenecektir.

Form tasarımlarımız bittiğine göre artık kodlamaya geçebiliriz.

Projemizi yanda bulunan resimde ki gibi hazırladıktan sonra Northwind veritabanının yansımısını alarak kodlarımızı yazmaya hazır hale gelebiliriz.

Form4 ekranı, iki adet büyük datagridview kontrolüne sahiptir. Üstteki datagridview tüm siparişleri listelerken, alttaki datagridview seçilen siparişin detaylarını gösterir.

Form1.cs

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    //Veritabanımızın instance'ını alıyoruz.
    NorthwindEntities db = new NorthwindEntities();
    private void Form1_Load(object sender, EventArgs e)
    {
        //Sorgularımızı yazarak çeşitli verileri ekrana yazdırıyoruz. Çeşitlendirebilirsiniz.
        lblToplamUrunSayisi.Text = db.Products.Count().ToString();
        lblStokSayisi.Text = db.Products.Sum(p => p.UnitsInStock).ToString();
        lblToplamSiparisSayisi.Text = db.Orders.Count().ToString();
    }

    private void ürünİşlemleriToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Form2 frm = new Form2();
        frm.ShowDialog();
    }

    private void siparişleriGörüntüleToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Form4 frm = new Form4();
        frm.ShowDialog();
    }
}
```

Form1 üzerinde yapılacak işlemler yukarıdaki gibidir. Form Load eventinde formun açılışında ekranda gösterilecek değerleri veritabanından getirdik. MenuStrip içerisindeki butonların açacağı formları da yazdık ve Form1'de işimiz bitti.

Şimdi ürünleri sepete ekleyebilmek için yeni bir class oluşturacağız. Classımızın adı SepetUrun olacaktır. Sepete eklenen ürünleri SepetUrun tipine dönüştürüp RAM üzerinde tutacağız.

SepetUrun.cs

```
public class SepetUrun
{
    //Sepetin içerisinde tutulacak ürün bilgisini Product tipinde tutuyoruz.
    public Product SepettekiUrun { get; set; }
    //Adet bilgisine üründen kaç adet sipariş verileceğini bilmemiz için gerek duyuyoruz.
    public int Adet { get; set; }

    //Class'ın ToString metodunu eziyoruz.
    public override string ToString()
    {
        return string.Format("{0} x{1}", SepettekiUrun.ProductName, Adet);
    }
}
```

```
}
```

SepetUrun classımızı oluşturduktan sonra Form2'yi kodlamaya başlayabiliriz.

Form2.cs

```
public partial class Form2 : Form
{
    public Form2()
    {
        InitializeComponent();

        //Veritabanı instance'ını alıyoruz.
        NorthwindEntities db = new NorthwindEntities();
        //Sepeti oluşturuyoruz. Sepeti oluşturmak için yukarıdaki SepetUrun.cs classını
        oluşturmalıyız.
        List<SepetUrun> mevcutListe = new List<SepetUrun>();

        private void Form2_Load(object sender, EventArgs e)
        {
            //Kategorileri comboBox içerisine listeliyoruz.
            cbKategorisi.DataSource = db.Categories.ToList();
            cbKategorisi.DisplayMember = "CategoryName";
            cbKategorisi.ValueMember = "CategoryID";

            //UrunleriListele Metodu oluşturup ürün listeleme işlemini yapıyoruz.
            UrunleriListele();

            //SepetDurumunuGoster metodu ekleyerek sepete eklenen ürünlerin listBox'a eklenmesini
            sağlıyoruz.
            SepetDurumunuGoster();
        }

        private void UrunleriListele()
        {
            //Veritabanındaki tüm ürünleri DataGridView kontrolüne listeleyen bir metod yazıyoruz.
            dgvUrunler.DataSource = db.Products.Select(p => new {
                UrunID = p.ProductID,
                UrunAdi = p.ProductName,
                UrunFiyati = p.UnitPrice,
                StokMiktari = p.UnitsInStock
            }).ToList();
        }

        private void btnUrunKaydet_Click(object sender, EventArgs e)
        {
            //Kaydet butonu içerisinde yeni ürün eklemek için gerekli kodlarımızı yazıyoruz.
            Product yeni = new Product();
            yeni.ProductName = txtUrunAdi.Text;
            yeni.UnitPrice = nmrFiyati.Value;
        }
    }
}
```

```
        yeni.UnitsInStock = Convert.ToInt16(nmrStokMiktari.Value);
        //yeni.Category null değer taşıdığından önce Category classından instance alıyoruz ve
        //değerlerimizi sonrasında oluşturulan yeni.Category nesnesine ekleyebiliriz.
        yeni.Category = new Category { CategoryID = Convert.ToInt32(cbKategorisi.SelectedValue)
    };

    db.Products.Add(yeni);
    bool sonuc = db.SaveChanges() > 0;
    if (sonuc)
    {
        UrunleriListele();
        MessageBox.Show("Başarıyla kayıt edildi");
    }
    else
    {
        MessageBox.Show("Ürün eklenirken bir hata oluştu");
    }
}

private void txtAra_TextChanged(object sender, EventArgs e)
{
    //Ürün Adında, txtAra textBox kontrolüne yazılan karakterleri içeren ürünleri
    //listeliyoruz.
    dgvUrunler.DataSource = db.Products.Where(p=>
    p.ProductName.Contains(txtAra.Text)).ToList();
}

private void dgvUrunler_DoubleClick(object sender, EventArgs e)
{
    //DataGridView içerisinde seçilen satır yoksa eventten çıkıyoruz.
    if (dgvUrunler.SelectedRows.Count < 1) return;

    //DataGridView'da seçili ürünün ID bilgisini yakalıyoruz.
    int seciliID = Convert.ToInt32(dgvUrunler.SelectedRows[0].Cells[0].Value);
    //Yakaladığımız ID'yi veritabanından Product nesnesi olarak getiriyoruz.
    Product sepeteEklenecek = db.Products.Find(seciliID);
    //Tıklanan ürün oluşturduğumuz sepetimizde varmı diye kontrol ediyoruz.
    bool sepetteVarmi = mevcutListe.Count > 0 && mevcutListe.FirstOrDefault(su =>
    su.SepettekiUrun.ProductID == seciliID) != null;

    if (sepetteVarmi)
    {
        //Sepette varsa; sepetteki varolan SepetUrunu ekde ediyoruz!
        SepetUrun varolan = mevcutListe.FirstOrDefault(su => su.SepettekiUrun.ProductID ==
        seciliID);

        //Stok miktarı izin veriyorsa sepetteki ürünün adedini bir arttırıyoruz.
        if (varolan.Adet < sepeteEklenecek.UnitsInStock)
        {
            varolan.Adet++;
        }
    }
}
```

```
        }
        else
        {
            MessageBox.Show("Yeterli stok kalmadı!");
        }
    }
    else
    {
        //Eğer ürün sepette yoksa sepete eklememiz gerekiyor. Sepete eklenmesi için ürün
        //stoğunun yeterli olması gerekmektedir.
        if (sepeteEklenecek.UnitsInStock > 0)
        {
            SepetUrun yeniUrun = new SepetUrun()
            {
                SepettekiUrun = sepeteEklenecek,
                Adet = 1
            };
            mevcutListe.Add(yeniUrun);
        }
        else
        {
            MessageBox.Show("Stok yetersiz.");
        }
    }
    SepetDurumunuGoster();
}

private void SepetDurumunuGoster()
{
    //Sepetin güncel halini listBox kontrolüne yazdırıyoruz.
    lstSepet.Items.Clear();

    if (mevcutListe.Count < 1)
    {
        lstSepet.Items.Add("Sepette ürün yok!");
    }
    else
    {
        foreach (SepetUrun item in mevcutListe)
        {
            lstSepet.Items.Add(item);
        }
    }
}

private void lnkSiparisVer_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    //Sepeti sipariş verme formuna gönderiyoruz.
    Form3 frm = new Form3(mevcutListe);
}
```



```
        frm.ShowDialog();  
    }  
}
```

Form2'de yapmamız gereken işler tamamlandığına göre artık Form3 kodlamasına geçebiliriz. Form'de oluşturduğumuz sepeti Form3'de yakalayıp daha sonra bir sipariş oluşturarak sepet ürünlerini siparişin detayları olarak eklememiz gerekmektedir.

Form3.cs

```
public partial class Form3 : Form  
{  
    public Form3()  
    {  
        InitializeComponent();  
    }  
  
    List<SepetUrun> mevcutListe;  
    NorthwindEntities db = new NorthwindEntities();  
  
    //Constructor üzerinden Form2'den gönderilen sepeti yakalıyoruz. Sepeti kullanabilmek için  
    global alana çıkartmamız gerekmektedir.  
    public Form3(List<SepetUrun> _mevcutListe)  
    {  
        InitializeComponent();  
        mevcutListe = _mevcutListe;  
    }  
  
    //Çalışanları, Kargocuları ve Müşterileri comboBox kontrollerine dolduruyoruz.  
    private void Form3_Load(object sender, EventArgs e)  
    {  
        cbCalisanlar.DataSource = db.Employees.Select(emp => new { EmployeeName = emp.FirstName  
+ " " + emp.LastName, CalisanID = emp.EmployeeID }).ToList();  
        cbCalisanlar.DisplayMember = "EmployeeName";  
        cbCalisanlar.ValueMember = "CalisanID";  
  
        cbMusteriler.DataSource = db.Customers.ToList();  
        cbMusteriler.DisplayMember = "CompanyName";  
        cbMusteriler.ValueMember = "CustomerID";  
  
        cbKargocular.DataSource = db.Shippers.ToList();  
        cbKargocular.DisplayMember = "CompanyName";  
        cbKargocular.ValueMember = "ShipperID";  
  
        //Eğer sepet boş değilse listBox içerisine sepet içeriğini dolduruyoruz.  
        if (mevcutListe != null)  
        {  
            foreach (SepetUrun item in mevcutListe)  
            {  
                lstSepetBilgisi.Items.Add(item);  
            }  
        }  
    }  
}
```

```
}  
}  
  
private void btnSiparisKaydet_Click(object sender, EventArgs e)  
{  
    //Yeni bir sipariş kayıt ediyoruz.  
    if (mevcutListe.Count < 1)  
    {  
        MessageBox.Show("Sepette ürün bulunmamaktadır.");  
        return;  
    }  
  
    //Sipariş oluşturup siparişin bilgilerini veriyoruz.  
    Order siparis = new Order {  
        CustomerID = cbMusteriler.SelectedValue.ToString(),  
        EmployeeID = Convert.ToInt32(cbCalisanlar.SelectedValue),  
        ShipVia = Convert.ToInt32(cbKargocular.SelectedValue)  
    };  
  
    //Siparişi veritabanı yansımasına ekliyoruz.  
    db.Orders.Add(siparis);  
  
    //Değişiklikleri Kaydet metoduyla kayıdı veritabanı üzerinde gerçekleştiriyoruz.  
    bool sonuc1 = db.SaveChanges() > 0;  
    //Sepet'de ki ürünlerden kaç tanesinin başarıyla kayıt edildiğini saydırıyoruz.  
    int sayi = 0;  
  
    //Oluşturduğumuz sepet içerisinde dönüyoruz ve sepetin her ürününü bir OrderDetail  
    olarak kullanarak önceki siparişimizin detayı olarak eklenmesini sağlıyoruz.  
    foreach (SepetUrun item in mevcutListe)  
    {  
        Order_Detail od = new Order_Detail();  
        od.OrderID = siparis.OrderID;  
        od.ProductID = item.SepettekiUrun.ProductID;  
        od.Quantity = Convert.ToInt16(item.Adet);  
        od.UnitPrice = item.SepettekiUrun.UnitPrice == null ? 0 :  
Convert.ToDecimal(item.SepettekiUrun.UnitPrice);  
        od.Discount = 0;  
  
        db.Order_Details.Add(od);  
        sayi++;  
    }  
  
    //Eğer sepetdeki sayı ile eklenen sayı birbirini tutuyorsa kayıtda sıkıntı olmadığı için  
    true gelecektir.  
    bool sonuc2 = lstSepetBilgisi.Items.Count == sayi;  
  
    //Siparişin ve Sipariş detaylarının sonucu true ise sipariş başarıyla eklenmiş demektir.  
    if (sonuc1 && sonuc2)  
    {
```

```
        MessageBox.Show("Başarıyla sipariş alındı!");
        db.SaveChanges();
    }
    else
    {
        MessageBox.Show("Sipariş detayları alınırken bir hata oluştu!");
    }
}
}
```

Form3'de ki sipariş ve sipariş detaylarını kayıt işlemi tamamlandığına göre artık Form4 üzerinde siparişlerin görüntülenmesini ayarlayabiliriz.

Form4.cs

```
public partial class Form4 : Form
{
    public Form4()
    {
        InitializeComponent();

        NorthwindEntities db = new NorthwindEntities();

        private void Form4_Load(object sender, EventArgs e)
        {
            //Tüm siparişleri listeliyoruz
            dgvSiparisler.DataSource = db.Orders.ToList();
        }

        private void dgvSiparisler_DoubleClick(object sender, EventArgs e)
        {
            //Eğer bir sipariş seçilmemişse eventten çık!
            if (dgvSiparisler.SelectedRows.Count < 1) return;

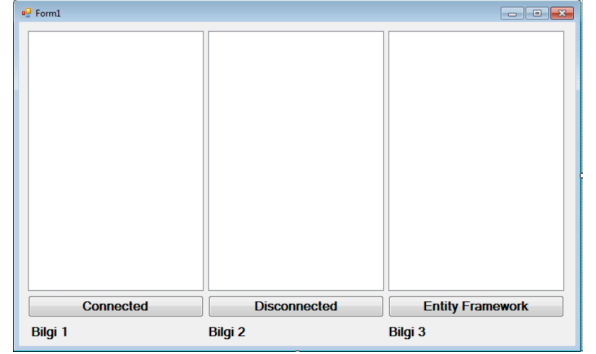
            //Çift tıklanan siparişin sipariş detaylarını listeliyoruz.
            int id = Convert.ToInt32(dgvSiparisler.SelectedRows[0].Cells[0].Value);
            dgvSiparisDetaylari.DataSource = db.Order_Details.Where(od => od.OrderID ==
id).ToList();
        }
    }
}
```

Entity Framework Performans Karşılaştırma

Connected, Disconnected ve Entity Framework performans kıyaslaması yapacağız. Bu 100%'lük bir kıyaslama olmayacaktır. Çünkü bu işlemi ayrı ayrı projelerde yapmamız gerekecektir. Ama yine de bize bir fikir verecektir.

Bu karşılaştırma için öncelikle sağda ki resimde bulunan formu tasarlayalım ve butonlar altına aşağıdaki komutları yazalım. Komutların çalışma süreleri label kontrollerinde görünecektir.

Connected mimari için kodlarımızı yazarak projemize başlayabiliriz.



Connected Mimari

```
DateTime baslangic = DateTime.Now;
SqlConnection cnn = new SqlConnection("Server=.;Database=Northwind;UID=sa;PWD=123");
SqlCommand cmd = new SqlCommand("Select * From Orders", cnn);
if (cnn.State == ConnectionState.Closed) cnn.Open();
SqlDataReader dr = cmd.ExecuteReader();
if (dr.HasRows)
{
    while (dr.Read())
    {
        lstConnected.Items.Add(dr["OrderID"]);
    }
}
cnn.Close();
DateTime bitis = DateTime.Now;
TimeSpan fark = bitis - baslangic;

lblConnected.Text = fark.Milliseconds.ToString();
```

Connected mimari her seferinde güncel veriyi adım adım veritabanından çekeceği için yavaş çalışacaktır.

Disconnected Mimari

```
DateTime baslangic = DateTime.Now;
SqlDataAdapter dap = new SqlDataAdapter("Select * From Orders", "Server=.; Database=Northwind; UID=sa; PWD=123");
DataTable dt = new DataTable();
dap.Fill(dt);
lstDisconnected.DataSource = dt;
lstDisconnected.DisplayMember = "OrderID";
```

```
DateTime bitis = DateTime.Now;  
TimeSpan fark = bitis - baslangic;  
lblDisconnected.Text = fark.Milliseconds.ToString();
```

Disconnected mimari her çalışmasında hızlı çalışacaktır. Bunun sebebi veriyi cache'den çekmesidir.

Entity Framework

```
private void btnEntity_Click(object sender, EventArgs e)  
{  
    NorthwindEntities db = new NorthwindEntities();  
    DateTime baslangic = DateTime.Now;  
    lstEntity.DataSource = db.Orders.ToList();  
    lstEntity.DisplayMember = "OrderID";  
    DateTime bitis = DateTime.Now;  
    TimeSpan fark = bitis - baslangic;  
    lblEntity.Text = fark.Milliseconds.ToString();  
}
```

Entity Framework ilk seferinde, yani yansıma Cache'e gelene kadar yavaş çalışır. Daha sonraki çalışmalarda entity framework hızlanacaktır.

Entity Framework'un instance'ı nı global'e taşıdığımızda proje derlenirken yansıma alınacağı için hızı artacaktır.

ENTITY FRAMEWORK – CODE FIRST

Kod ile daha fazla işleme müdahale ederek, SQL tarafında olmayan bir veritabanını oluşturmak için kullanılır. Bir programı birden fazla sisteme kurmak için kolaylık sağlar. Program içerisinde veritabanının oluşmasını sağlar. Bunun dışında değişiklik yapıldığında veritabanında da bu değişiklikleri yapmayı sağlayabiliriz.

Code First yaklaşımını kullanarak bir veritabanı oluşturma işlemi yapalım. Bu işlem için öncelikle "Windows Forms Application" projesi içerisinde aşağıda ki classları oluşturalım.

Category.cs

```
public class Category  
{  
    public int CategoryID { get; set; }  
    public string CategoryName { get; set; }  
    public string Description { get; set; }  
}
```

Category classımızı oluşturduktan sonra Product.cs isimli bir class daha oluşturuyoruz ve aşağıdaki property tanımlamalarını yapıyoruz.

Product.cs

```
public class Product
{
    public int ProductID { get; set; }
    public string ProductName { get; set; }
    public int UnitsInStock { get; set; }
    public decimal UnitPrice { get; set; }
    public int CategoryID { get; set; }
}
```

Yukarıda oluşturduğumuz classlar bizim veritabanında bulunan tablolarımızın classları olacaktır. Code First yaklaşımında oluşturduğumuz classları kullanarak veritabanı oluşturulur.

Database First yaklaşımında, Entities isimli bir class bulunuyordu.. Bu class bizim tablolarımızın bilgilerini içerisinde tutmaktaydı. Database First yaklaşımında Entities classını T4 sistemini kullanarak kendi oluşuyordu fakat Code First yaklaşımında bu classı kendimiz yazmalıyız. Bunun için aşağıdaki NorthwindShopEntities isimli classı oluşturalım.

Entity Framework mimarisinin güzelliği burada devreye girer. NorthwindShopEntities classını doğru bir biçimde şekillendirdiğimizde içerisine verdiğimiz classlar veritabanında düzgün bir şekilde oluşturulabilir.

Bu işlemi yapmak için projemize "Entity Framework.dll" yüklememiz gerekiyor. Miras aldığımız DbContext classı Entity Framework kütüphanesinde yer almaktadır.

Aşağıda oluşturduğumuz class içerisinde istediğimiz yapıyı kullanabilmek için, classımızı diğer classlardan ayırmamız gerekmektedir. Bunun için "NorthwindShopEntities" classımıza "DbContext" classımızdan miras almamız gerekmektedir. **"DbContext", "System.Data.Entity" namespace altında bulunmaktadır.**

NorthwindShopEntities.cs

```
public class NorthwindShopEntities : DbContext //Miras Aldık
{
}
```

Daha sonra oluşturduğumuz "NorthwindShopEntities" classımız içerisinde property'leri belirtmemiz gerekiyor.

NorthwindShopEntities

```
public class NorthwindShopEntities : DbContext //Miras Aldık
{
    /* Syntax */
    /* DbSet<KolonlarHangiClassaGoreOlusturulacak> <TabloAdiNeOlacak> { get; set; } */
    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }
}
```

"NorthwindShopEntities.cs" classımızı yukarıdaki şekilde düzenliyoruz. Şimdi yukarıda bulunan classı tetikleterek veritabanımızı oluşturmamızı sağlayabiliriz. Formumuzun Load eventinde aşağıdaki kodları yazarak veritabanımızı oluşturabiliriz.

Form1.cs

```
private void Form1_Load(object sender, EventArgs e)
{
    NorthwindShopEntities db = new NorthwindShopEntities();
    var result = db.Categories.ToList();
}
```

}

Yukarıdaki işlem aslında bir veri çekmek işlemidir ama Code First yaklaşımında amacımız ilk çalıştırmada kategorileri çekmek değil, tablolarımızın oluşturulmasını sağlamaktır. Projeyi çalıştırdığımızda kısa bir beklemeden sonra form açılacaktır. Şimdi Sql Server Management Studio'yu açıyoruz. NorthwindShop isimli bir veritabanı oluşup oluşmadığını kontrol ediyoruz. Veritabanının oluşmadığını görüyoruz. Bunun sebebi ne olabilir?

Tabiki de sebebi bu yapıyı veritabanına oluşturabilmesi için gerekli ConnectionString tanımlamasını yapmadık. Bu sebeple arka tarafta bu işlemi localdb olarak oluşturdu. Biz bir connection string tanımlayıp bu işlemi veritabanı üzerinde yaptırılmaz gerekmektedir.

Bu işlem için ilk adım "App.config" içerisinde bir ConnectionString tanımlaması yapmalıyız. App.config içerisinde ConnectionStrings içerisinde aşağıdaki bağlantıyı tanımlıyoruz. ConnectionString içerisinde verdiğimiz veritabanı ismi ile Database oluşturulacaktır. Bizim database oluşturma işlemini Sql Management Studio tarafında yapmamıza gerek yoktur.

*ConnectionString oluştururken dikkat etmemiz gereken çok önemli bir nokta daha var. ConnectionString tanımlamasını ConfigSection'dan önce **yapmamamız** gerekiyor. **Config Section ve startup taglerinden sonra connection string tanımlamalıyız.***

Oluşturmanız gereken connection string tanımlaması için bir örnek aşağıda yer almaktadır. Örneğimizde aşağıdaki connectionString üzerinden işlemleri gerçekleştiriyor olacağız.

App.config

```
<add name="NorthwindShopDBConn" connectionString="Server=ALP1-BILGISAYAR\EMPATI;
Database=NorthwindShopDB; UID=sa; PWD=123" providerName="System.Data.SqlClient" />
```

Appconfig içerisinde aşağıda bulunan siyah renk ile yazılmış, altı çizili satırı silelim.

App.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
    http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
    type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
    Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <connectionStrings>
    <add name="NorthwindShopDBCon" connectionString="Server=ALP1-BILGISAYAR\EMPATI;
    Database=NorthwindShopDB; UID=sa; PWD=123" providerName="System.Data.SqlClient" />
  </connectionStrings>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory,
    EntityFramework" />
    <providers>
      <provider invariantName="System.Data.SqlClient"
      type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
</configuration>
```

Yukarıda siyah ile işaretlenmiş satırları silebiliriz. Siyah ile işaretlenmiş tanımlama, Entity Framework tarafından oluşturulmuş bir ifadedir. Bu işlemden sonra bağlantımızıda oluşturduğumuza göre artık “NorthwindShopEntities.cs” classı içerisinde gerekli düzenlemeyi yaparak veritabanı oluşturma işleminin local’de değilde, veritabanında oluşmasını sağlayabiliriz.

Öncelikle NorthwindShopEntities classının Constructor metodunu oluşturalım.

NorthwindShopEntities.cs

```
public class NorthwindShopEntities : DbContext //Miras Aldık
{
    //Constructor Metot
    public NorthwindShopEntities()
    {
    }

    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }
}
```

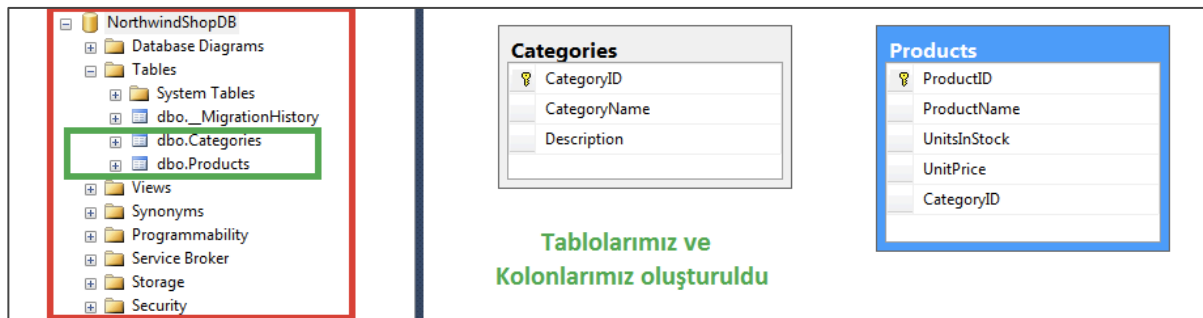
DbContext’den miras almıştık. Şimdi bu mirası kullanarak veritabanımıza oluşturma işlemlerimizi gerçekleştiriyoruz. Bu işlem için base classımızın constructor’ına (DbContext) connection string göndermemiz gerekiyor. Bu işlem için aşağıdaki kodları uygulayalım.

NorthwindShopEntities.cs

```
public class NorthwindShopEntities : DbContext //Miras Aldık
{
    public NorthwindShopEntities():base("NorthwindShopDbConn") //Buradaki bağlantıyı kullanacak
    base anahtar kelimesi ile bağlantı adını, base classın constructor’ına gönderiyor.
    {
    }

    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }
}
```

Yukarıda DbContext classının hangi connection string’i kullanacağını belirtiyoruz. Verdiğimiz ConnectionString üzerinden veritabanına bağlantıları oluşturacaktır. Sql Server Management Studio’yu arayüzümüzü açıp veritabanımızı kontrol edelim.



Yukarıdaki resimde gördüğünüz gibi ConnectionString’de belirttiğimiz veritabanı ismi ile veritabanımız, “NorthwindShopEntities.cs” classımızda belirttiğimiz tablo isimleri ile veritabanı tablolarımız oluşturuldu. Veritabanımız içerisinde bulunan tabloların kolonları, classlarımız (Product, Category) içerisinde bulunan propertyler ile belirlendi. Ama yukarıdaki tabloların görünümünü Standart olarak kolon tiplerinin otomatik olarak belirlendiğini görürüz.

Örneğin kategori adı kolonumuz nvarchar(MAX) olarak belirlenmiştir. Aynı şekilde Products tablosunda ki UnitPrice kolonu decimal olarak belirlenmiştir. Ayrıca string tipte olan kolonlar boş geçilebilir olarak işaretlenmiştir. Bunun dışında Propertylerde ilk yazılan property Uniq kolon olarak seçilmiştir. Bu bilgilerin ardından diagrama bir göz atalım.

Aşağıdaki resimde diagramı görebilirsiniz.

Categories		
Column Name	Data Type	Allow Nulls
CategoryID	int	<input type="checkbox"/>
CategoryName	nvarchar(MAX)	<input checked="" type="checkbox"/>
Description	nvarchar(MAX)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Products		
Column Name	Data Type	Allow Nulls
ProductID	int	<input type="checkbox"/>
ProductName	nvarchar(MAX)	<input checked="" type="checkbox"/>
UnitsInStock	int	<input type="checkbox"/>
UnitPrice	decimal(18, 2)	<input type="checkbox"/>
CategoryID	int	<input type="checkbox"/>
		<input type="checkbox"/>

Yukarıdaki bilgilere el ile müdahale edebilmemiz gerekmektedir. Bunun için C sharp tarafında bu bilgilere el ile müdahale edebileceğimiz bir yer var. Model oluşturulurken, propertyler ile ilgili ayarları yapabiliriz. Bu yer benim için nesneyi override ettiğim yerdir. "NorthwindShopEntities.cs" içerisinde model oluşturulurken override işlemi gerçekleştirerek istediğim özellikleri ayarlayabilirim.

Ezmek istediğim yer OnModelCreating yani modelin oluşturulduğu andır. Burada modelimi yakalar ve içerisinde istediğim değişikliği yapabiliriz.

Aşağıdaki kodları düzenledikten sonra çalıştırmadan önce, Sql Server Management Studio'da bulunan NorthwindShopDB veritabanını silelim. Daha sonra aşağıdaki kodu çalıştıralım.

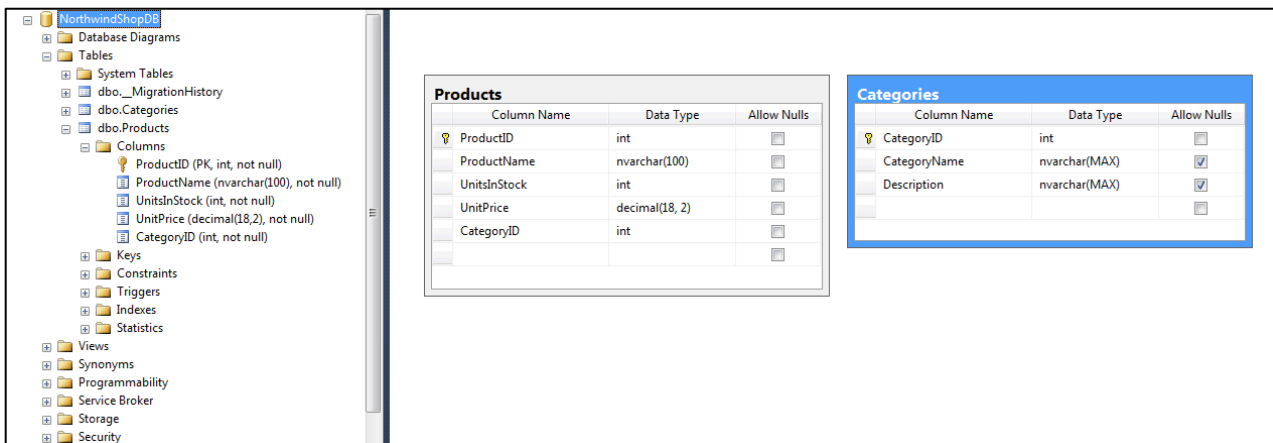
NorthwindDbEntities.cs

```
//Model oluşturulurken
protected override void OnModelCreating(DbModelBuilder modelBuilder) //Değer ModelBuilder'da yakalanır
{
    //Ürün adı boş geçilemez, uzunluğu 100 karakter olsun
    //Model Builder içerisinde Product değerimizi yakaladık. Property'leri içerisinde
    ProductName'in maxLenght bilgisini belirledik ve boş geçilemez olarak ayarladık.
    modelBuilder.Entity<Product>().Property(p => p.ProductName).HasMaxLength(100).IsRequired();
}
```

IsRequired => Boş Geçilemez.

HasMaxLength => Maximum karakter sayısı

Yukarıdaki işlemten sonra veritabanımızı kontrol edelim. Kolonumuzun özellikleri max length değeri 100 ve boş geçilemez olarak düzenlenmiş olacaktır.



Products		
Column Name	Data Type	Allow Nulls
ProductID	int	<input type="checkbox"/>
ProductName	nvarchar(100)	<input type="checkbox"/>
UnitsInStock	int	<input type="checkbox"/>
UnitPrice	decimal(18, 2)	<input type="checkbox"/>
CategoryID	int	<input type="checkbox"/>
		<input type="checkbox"/>

Categories		
Column Name	Data Type	Allow Nulls
CategoryID	int	<input type="checkbox"/>
CategoryName	nvarchar(MAX)	<input checked="" type="checkbox"/>
Description	nvarchar(MAX)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Yukarıda görüldüğü üzere veritabanımızda bulunan Products tablosunun ProductName kolonunun DataType'ı 100 olarak düzenlenmiş ve boş geçilemez olmuştur.

Peki biz veritabanı oluşturma işlemi sırasında her seferinde veritabanını Sql Server Management Studio'dan silecekmiyiz? Tabi ki de hayır. Biz bu işlem için bir strateji belirleyerek çalışabiliriz. Oluşturacağımız strateji biz projemizi her çalıştırdığımızda Veritabanını silecek ve tekrardan oluşturacaktır.

NorthwindDbEntities.cs

```
//Model oluşturulurken
protected override void OnModelCreating(DbModelBuilder modelBuilder) //Değer ModelBuilder'da
yakalanır
{
    //Stratejimizi belirleyelim. (Model değiştiğinde, veritabanını sil ve modeli tekrar oluştur)
    Database.SetInitializer<NorthwindShopEntities>(new
DropCreateDatabaseIfModelChanges<NorthwindShopEntities>());

    //Ürün adı boş geçilemez, uzunluğu 100 karakter olsun
    //Model Builder içerisinde Product değerimizi yakaladık. Property'leri içerisinde
ProductName'ın maxLenght bilgisini belirledik ve boş geçilemez olarak ayarladık.
    modelBuilder.Entity<Product>().Property(p => p.ProductName).HasMaxLength(100).IsRequired();
}
```

Tablolar Arası Bağlantıyı Ayarlamak

Şu ana kadar modellerimizi, veritabanımızı, tablolarımızı ve kolonlarımızı oluşturduk ama yaptığımız işlemlerde bir eksik var. Bu eksik tablolar arası bağlantıların kurulmamış olmasıdır. Oluşturduğumuz modeller üzerinden tablolarımız arasında bağlantılarımızı kurabiliriz.

Product ve Category tablomuzda bire çok bir bağlantı söz konusudur. Bizim iki tablo için bire çok bağlantıyı uygulayabilmemiz gerekiyor. Bu işlem için modellerimiz içerisinde bu bağlantıyı tanımlamalıyız. Bu işleme 'Mapping' adı verilir.

Mapping Uygulamak

Mapping işlemi tablolarımız arasında bağlantı kurmamızı sağlar. Classlarımızı aşağıda ki gibi düzenlediğimiz de Mapping işlemi tamamlanmış oluruz. Burada dikkat etmemiz gereken nokta, veritabanındaki normalizasyon kurallarımızdır. Northwind veritabanı üzerinden hareket ettiğimizde aşağıdaki sonuca ulaşırız.

- Bir ürünün, bir kategorisi vardır.
- Bir kategoriye ait birden fazla ürün vardır.

Yukarıdaki bilgilere göre bağlantımızı kurmamız gerekmektedir. Tablolar arası bağlantıyı hem 'Product.cs', hemde 'Category.cs' classı içerisinde tanımlamalıyız.

Product.cs

```
public class Product
{
    public int ProductID { get; set; }
    public string ProductName { get; set; }
    public int UnitsInStock { get; set; }
    public decimal UnitPrice { get; set; }
    public int CategoryID { get; set; }
    public Category Category { get; set; } //Mapping
}
```

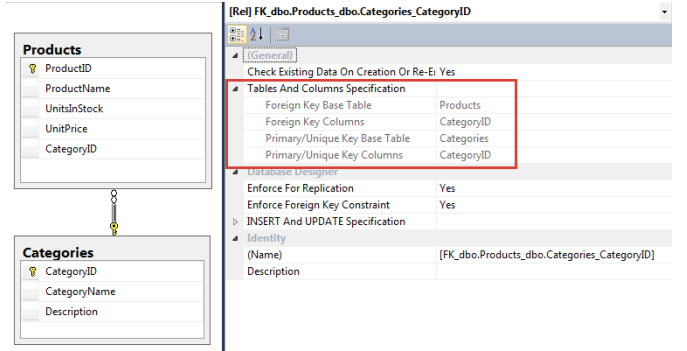
Category.cs

```
public class Category
{
    public int CategoryID { get; set; }
    public string CategoryName { get; set; }
    public string Description { get; set; }

    //Mapping
    public List<Product> Products { get; set; }
}
```

Product.cs içerisine eklediğimiz Category tipindeki property bu classların bağlantısını sağlamak içindir. Bu property her ürünün bir kategorisi olduğu anlamına gelir. Aynı zamanda Category.cs içerisine eklediğimiz List<Product> tipindeki property ise bir kategoriye ait birden fazla ürün olacağı anlamına gelecektir.

Yanda ki resimde gördüğünüz gibi tablolarımız arasındaki bağlantı oluşmuştur. Peki Categories tablosunda bulunan CategoryID ile Products tablosunda bulunan CategoryID'yi bağlayacağını nasıl anladı?



```
namespace WFACodeFirst
{
    public class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsInStock { get; set; }
        public decimal UnitPrice { get; set; }
        public int CategoryID { get; set; }

        //Mapping
        public Category Category { get; set; }
    }
}
```

EŞLEŞTİRİLDİ!

Tabiki de 'Mapping' olarak belirlediğim Category tipimi, Product propertyleri içerisindeki CategoryID ile eşleştirdi. Eğer biz CategoryID isminde bir property oluşturmasaydık bu kolonu tabloları oluşturma sırasında kendi oluşturacaktır.

Not : MigrationHistory : SQL'e aktarılırken tutulan bilgiler tablosudur. Entity Framework 5'e kadar el ile oluşturulurdu. Default olarak oluşturulacak bilgiler tanımlanırdı. Yeni versiyonlarda otomatik olarak oluşturuluyor.

Primary Key Belirlemek

Oluşturduğumu modeller üzerinde PrimaryKey kolonunu belirleyebiliriz. Primary Key kolonunu iki yöntem ile belirleyebiliyoruz. Birinci yöntem Primary Key kolonunu bir attribute ile işaretlemektir. İkinci yöntem ise property ismi içerisinde 'ID' tanımlamasını kullanmaktır. Aşağıda iki yöntemide inceleyebiliriz.

1.Yöntem – KEY Attribute’u Kullanmak

Attribute : Key

```
[Key]
public int CategoryID { get; set; } //PK Kolonu
public string CategoryName { get; set; }
public string Description { get; set; }
```

Yukarıda Primary Key olarak belirlemek istediğimiz kolonu [Key] ile işaretliyoruz. Bu işaretleme belirtilen property’nin PK kolonu olacağını belirler. PK olarak belirlenmiş kolon otomatik olarak 1’den başlar ve 1 artar şeklinde belirlenecektir.

2.Yöntem – ID Eki

2.Yöntem – ID Eki

```
public int CategoryID { get; set; } //PK Kolonu
```

Oluşturduğumuz bir property sonuna ID eklediğimizde bu kolon PK kolonu olarak algılanacaktır.

ModelBuilder İşlemleri

Yukarıda oluşturduğumuz yapıyı geliştirmeye devam ediyoruz. Önceki kodlarımızda oluşturduğumuz veritabanında ki kolonları istediğimiz gibi düzenlemeliyiz. Örneğin UnitPrice kolonu ‘decimal’ oluşturulurken, aslında bu kolon ‘money’ tipinde oluşturulmalıydı.

Sıradaki işlemimiz veritipini değiştirmek olacaktır. Bu işlem için ‘NorthwindShopEntities.cs’ dosyasını açalım. Dosyamızı açtıktan sonra aşağıda ki kod’u projemize ekleyelim.

Kolon Tipini Değiştirmek

```
modelBuilder.Entity<Product>().Property(p => p.UnitPrice).HasColumnType("money");
```

HasColumnType ile kolon tipini belirleyebiliriz. Burada dikkat edilmesi gereken nokta string olarak verilecek kolon tipi Sql’de bulunan kolon tipleri ile uyumlu olmak zorundadır. Varsayılan olarak verilen IsRequired ve IsOptional değerler, belirtilen kolon tipine göre ayarlanır. Müdahale edilmediğinde Reference Type ve Value Typelara göre değişiklik gösterir. Reference Type’lar null değer alabileceğinden veritabanı tarafından null geçilebilir olarak algılanmaktadır. Yukarıdaki kodu ekledikten sonra classımız aşağıdaki gibi görünecektir.

NorthwindShopEntities.cs

```
public class NorthwindShopEntities : DbContext
{
    public NorthwindShopEntities() : base("NorthwindShopDBCon")
    {
    }

    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        Database.SetInitializer<NorthwindShopEntities>(new
        DropCreateDatabaseIfModelChanges<NorthwindShopEntities>());

        modelBuilder.Entity<Product>().Property(p =>
```

```
p.ProductName).HasMaxLength(100).IsRequired();

//Aşağıdaki satır ile veritipini money tipi yapıyoruz.
modelBuilder.Entity<Product>().Property(p => p.UnitPrice).HasColumnType("money");
}
}
```

UnitsInStock int tipinde, Required olarak aktarılabilecek bir property'dir. Bu sebeple UnitsInStock bilgisini burada belirtmemize gerek yoktur.

Categories tablosunda CategoryName boş geçilemez olması gerektiğinden bu kolonu yönetmemiz gerekir. Ayrıca CategoryName için maximum karakter sayısını belirlemeliyiz. Aşağıda bulunan iki satırı 'NorthwindShopEntities.cs' dosyamıza ekleyelim.

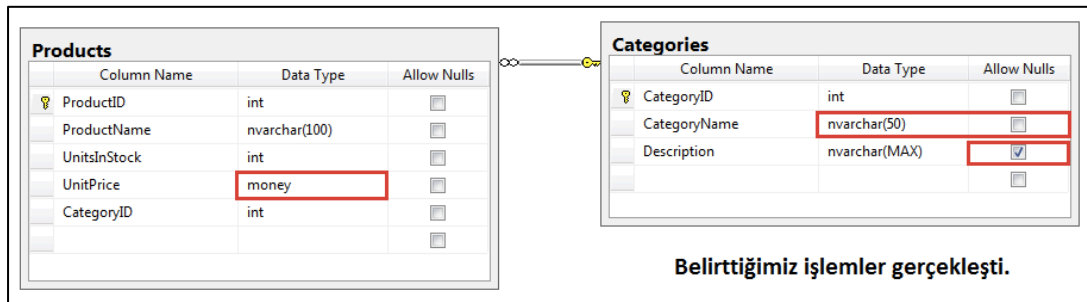
Zorunlu Kolon, Maximum Karakter Sayısı, Opsiyonel Alan Belirlemek

```
modelBuilder.Entity<Category>().Property(c=> c.CategoryName).IsRequired().HasMaxLength(50);
modelBuilder.Entity<Category>().Property(c=> c.Description).IsOptional();
```

Yukarıda CategoryName kolonunu IsRequired komutu ile boş geçilemez olarak belirledik. HasMaxLength komutu ile CategoryName kolonunun sınırını 50 karakter olarak belirledik.

Description kolonunu IsOptional, yani boş geçilebilir olarak belirledik. IsOption komutunu kullanmasaydım varsayılan olarak bu kolon boş geçilebilir bırakılacaktır.

Yukarıdaki yeni kodları 'NorthwindShopEntities.cs' dosyası içerisine ekledikten sonra projeyi çalıştıralım ve veritabanımızı inceleyelim.



Görüldüğü üzere modeli ezmemize bağlı olarak istediğimiz özellikler belirlenmiştir. **Dikkat etmemiz gereken bir nokta veritabanı üzerinde bir tablo veya diagram açıksa değişiklikleri uygulayamayabiliyor.** Bu sebepten projeyi tekrar çalıştırırken veritabanına ait tüm pencerelerin kapalı olduğuna emin olmalıyız.

Sırada yine çok kullanışlı olacak bir özellik var. Class içerisinde, veritabanında oluşturulmasını istemediğim bir property eklemek istediğimizde ne yapacağız? Sadece kullanmak için oluşturacağımız bu property'i aşağıdaki gibi yönetebiliriz. Örneğin; biz Description kolonunun veritabanında oluşturulmasını istemiyoruz.

Kolonun veritabanında oluşmasını engellemek

```
modelBuilder.Entity<Category>().Ignore(c => c.Description);
```

Yukarıdaki satırı projenize eklediğinizde Description kolonu oluşmayacaktır. Bir property sadece kullanılmak için oluşturulduğunda veritabanına oluşturulmasına gerek yoktur.

Kolonları oluşturma işlemlerini inceledik, şimdide propertylerimiz ile veritabanında bulunan kolon isimlerini farklı kullanmak istediğimizde, nasıl isim belirleneceğini inceleyelim. Aşağıdaki kod satırı bir property kolon olarak oluşturulurken ismini farklı vermek için kullanılmaktadır.

Kolon Adını Değiştirmek

```
modelBuilder.Entity<Product>().Property(p => p.ProductName).HasColumnName("UrunAdi");
```

Farklı isim kullanma işlemini tablolar içinde yapabiliriz.

Tablo Adını Değiştirmek

```
modelBuilder.Entity<Product>().ToTable("Urunler");
```

Code First Projesinde Entity Framework ile Ekleme İşlemi

Database First kullanırken öğrendiğiniz tüm sorguları Code First ile hazırladığınız veritabanında da kullanabilirsiniz. Aşağıda oluşturduğumuz veritabanı üzerine yeni bir kategori kayıt eden sorgu yer almaktadır.

Code First ile oluşturulan veritabanına ekleme işlemi yapmak

```
NorthwindShopEntities db = new NorthwindShopEntities();  
private void btnKaydet_Click(object sender, EventArgs e)  
{  
    Category c = new Category();  
    c.CategoryName = "Hello World";  
    db.Categories.Add(c);  
    db.SaveChanges();  
}
```