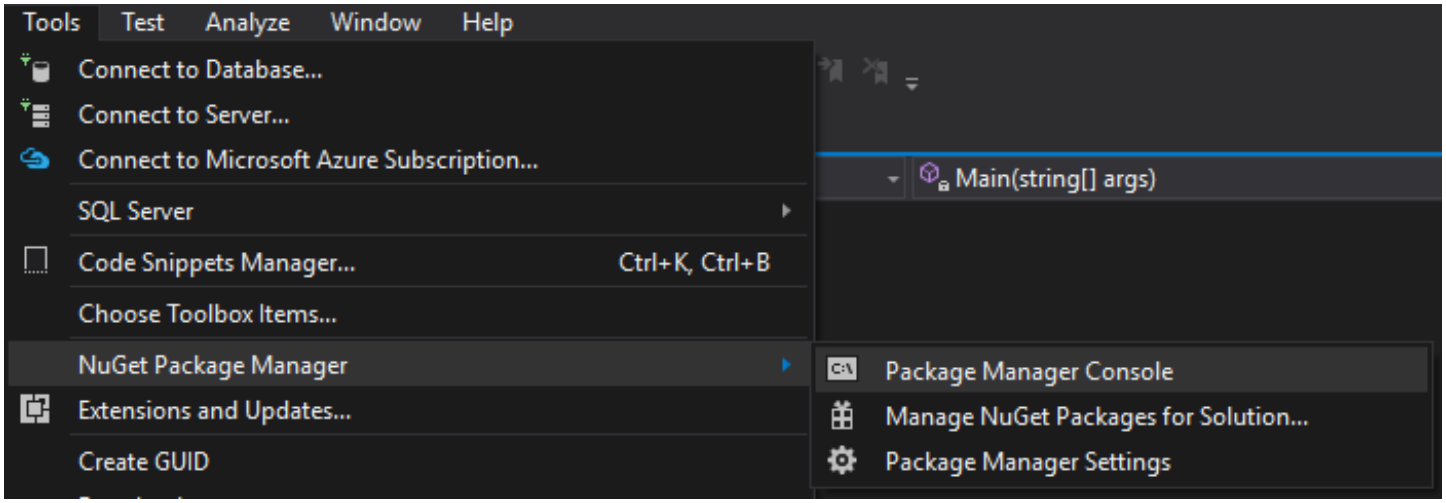
MIGRATIONS

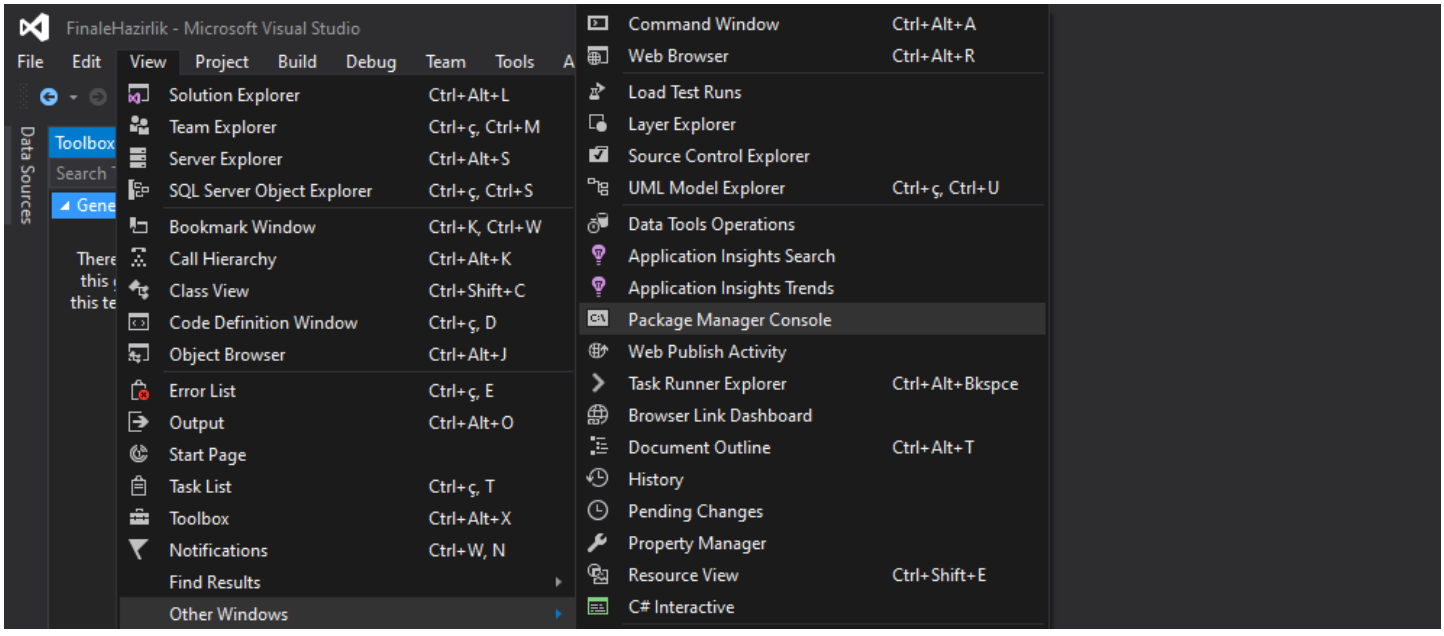
Code First deseni uygulanan bir projede veritabanına direkt olarak müdahale etmek oldukça sakıncalıdır. Haliyle yapacağımız ekleme, çıkarma veyahut güncelleme durumlarını tabloları temsil eden sınıflar (class) üzerinde gerçekleştirmeli ve güncellemeleri buradan gerçekleştirmeliyiz.

Şu ana kadar yaptığımız tüm işlemleri SQL Server' dan fiziksel veritabanını silip, yeniden yükleyerek gerçekleştiriyorduk. Bu veri kayıplarına neden oluyordu. İşte, Migration yapıları sayesinde yaptığımız yenilikleri Visual Studio üzerinden hızlıca fiziksel veritabanına yansıtabileceğiz. Anlayacağınız, **kod kısmında yaptığımız değişiklikleri veritabanına yansıtmaya Migration demekteyiz.**

Migrations yapısını kullanabilmek için bize **"Package Manager Console"** penceresi gerekmektedir. Bu pencereye ulaşmak için **"Tools" -> "NuGet Package Manager" -> "Package Manager Console"** kombinasyonunu takip edebilirsiniz.



Veya "View" -> "Other Window" -> "Package Manager" kombinasyonundan da aynı ekrana erişilebilir.



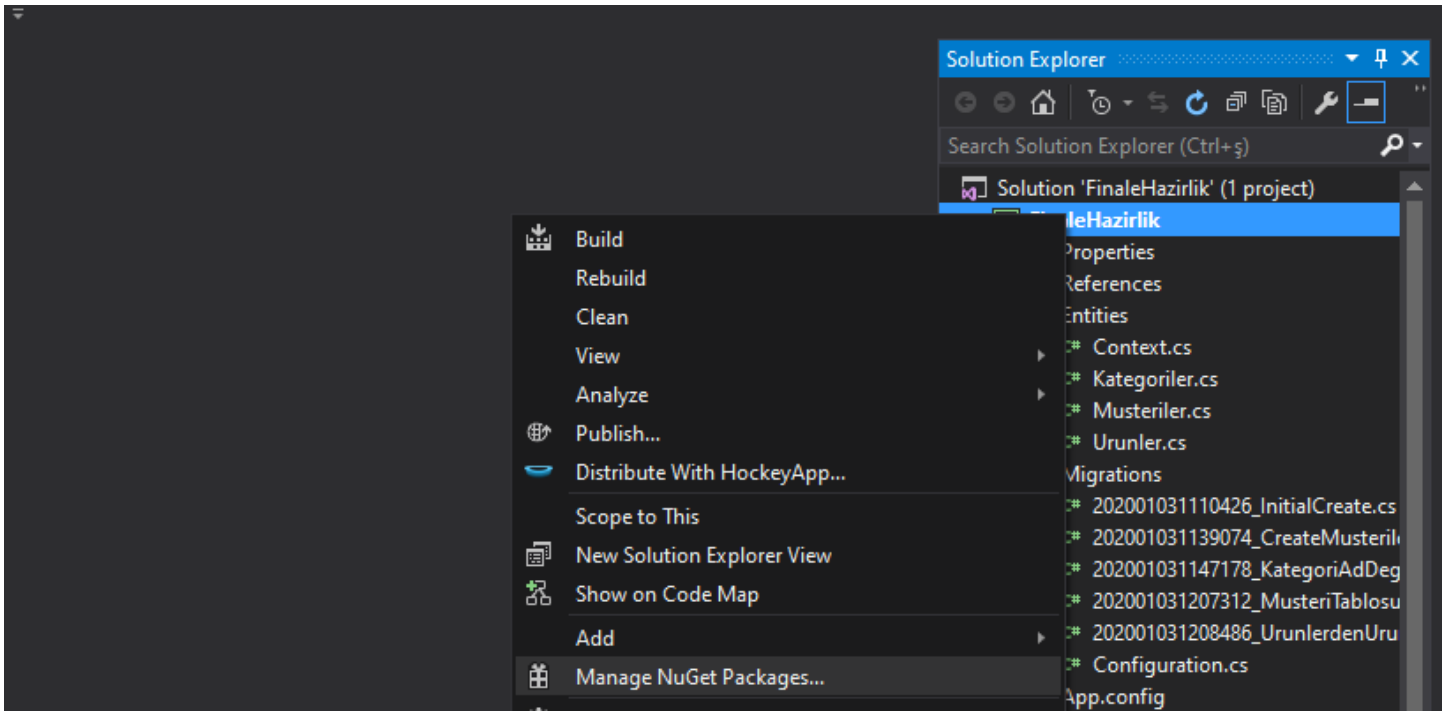
Visual Studio ile Code First deseni ile oluşturulan projelerde migration özelliği kapalı olarak gelir. Visual Studio üzerinden veritabanı üzerinde gerekli değişiklikleri yapabilmek için öncelikle bu özelliği aktif etmemiz gerekmektedir.

Basit bir proje üzerinde bu özellikleri gösterelim. Yapılacak işlemleri kısaca özetleyecek olursak

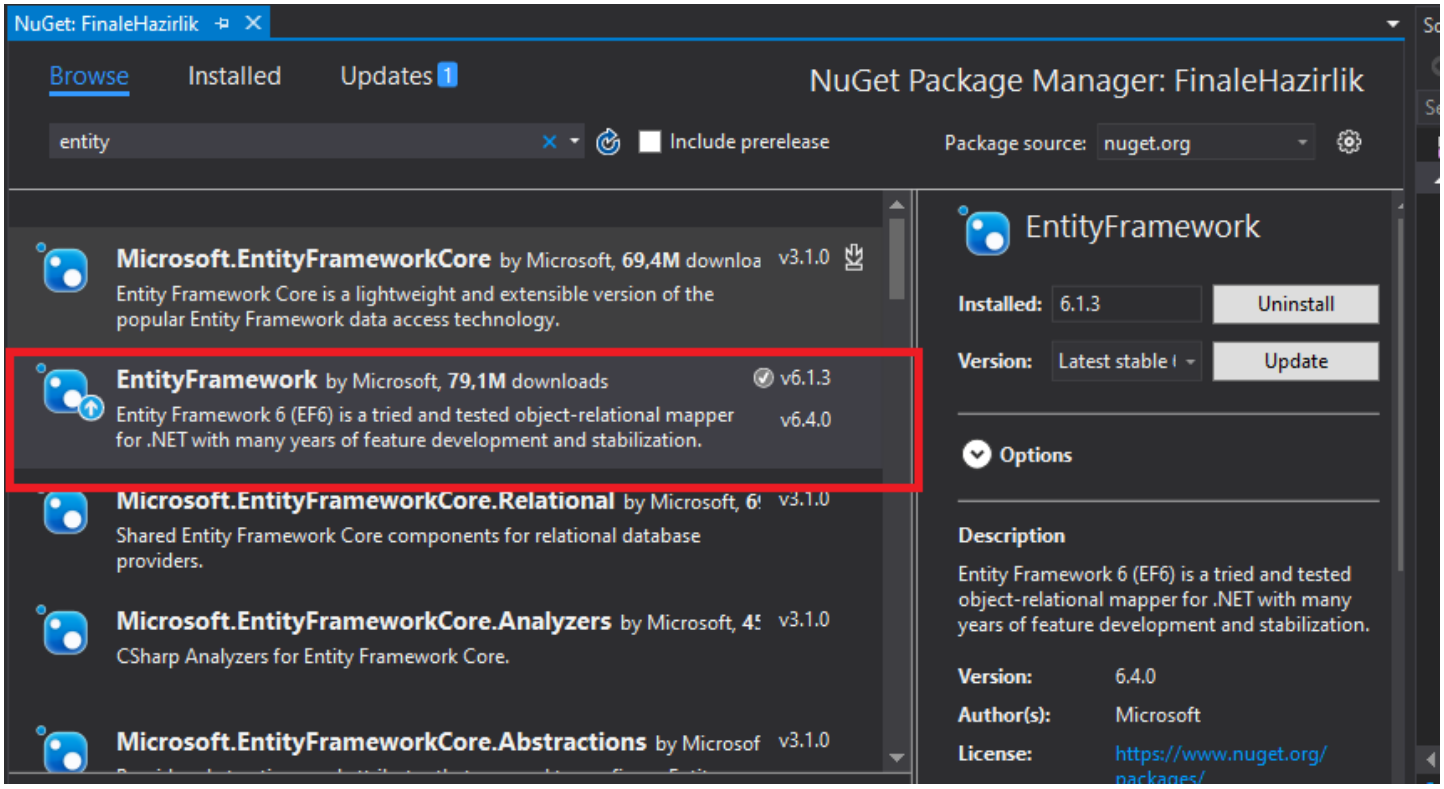
1. Boş projeyi oluştur.
2. Entity Framework ü projeye dahil et.
3. Entity klasörünü oluştur.
4. Tabloları oluşturacak sınıfları oluştur.
5. Tablolar üzerinde işlem yapılacak context sınıfını oluştur.
6. Veritabanı üzerinde yapılacak değişiklikler için Migrations oluştur.

1. Satış yapan bir firmanın ürünleri ile ilgili bilgileri tutacak bir veritabanı oluşturalım. Bunun için boş bir proje oluşturalım. Bunun için **“File” -> “New” -> “Project”** yolunu takip ederek yeni bir **“Windows Form Uygulaması”** oluşturalım.

2. **“Solition Explorer”** panelindeki proje klasörüne sağ tuşla tıklayarak açılan menüden **“Manage NuGet Packages”** yolunu kullanabilirsiniz.



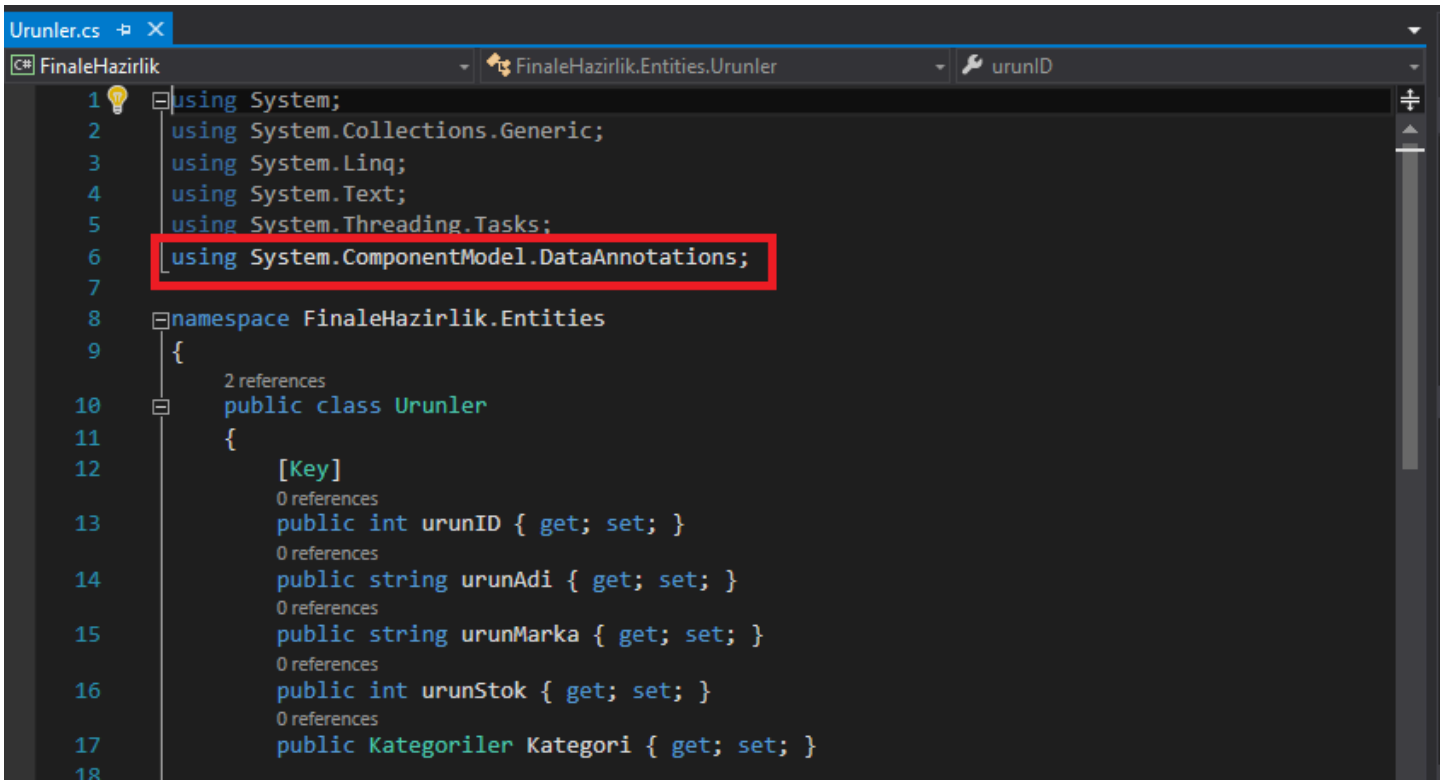
“NuGet Packages” panelinden **“EntityFramework”** seçerek sağ tarafta yer alan **INSTALL** butonunu kullanarak projenize ekleyebilirsiniz.



3. **“Solition Explorer”** panelindeki proje klasörüne sağ tuşla tıklayarak açılan menüden sırasıyla **“Add” -> “New Folder”** yolunu takip ederek projeye **“Entities”** klasörünü ekleyebilirsiniz.

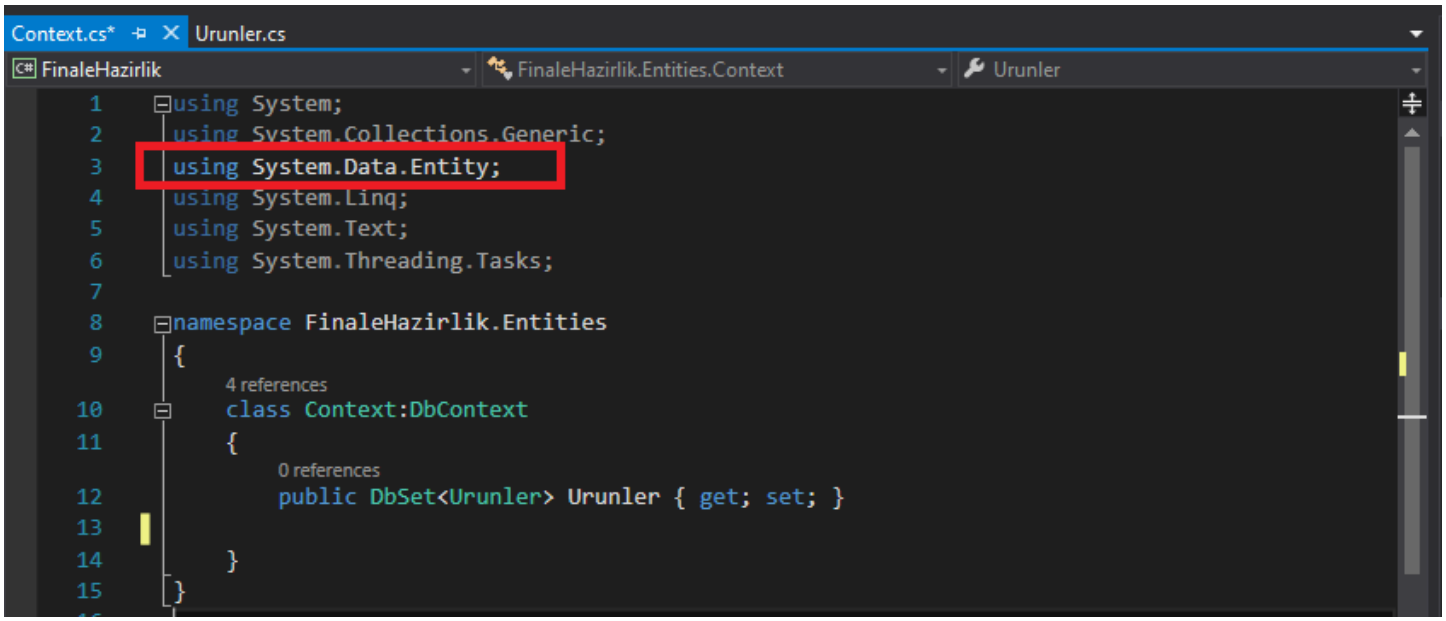
4. **“Solition Explorer”** panelindeki **“Entities”** klasörüne sağ tuşla tıklayarak açılan menüden sırasıyla **“Add” -> “Class”** yolunu takip ederek ürün bilgilerini tutacak tabloyu oluşturmak için **“Urunler.cs”** sınıfını aşağıdaki özellikleri tanımlayarak oluşturabilirsiniz.

***** Entity Framework Code First deseni kullanan uygulamalarda sınıf içerisinde tanımlama yapılan **ilk alanı otomatik olarak birincil anahtar (primary key) olarak ekler**. Bunun için ilgili sınıfın ilk alanının **ID olarak tanımlanması gerekir**. Aşağıdaki resimde kullanıldığı gibi ID isminden farklı bir alanı birincil anahtar olarak kullanabilmek için üstüne **[Key]** anahtarı kullanılmalıdır. [Key] anahtarının kullanılabilmesi için sınıfa **“System.ComponentModel.DataAnnotations”** kütüphanesinin eklenmesi gerekir.



5. **“Solition Explorer”** panelindeki **“Entities”** klasörüne sağ tuşla tıklayarak açılan menüden sırasıyla **“Add” -> “Class”** yolunu takip ederek veritabanı üzerinde işlem yapmamızı sağlayan context sınıfını aşağıdaki gibi tanımlayabiliriz.

Context sınıfını **“DbContext”** sınıfından türetebilmek için sınıf tanımlamalarına **“System.Data.Entity”** kütüphanesini eklemek gereklidir.



Tüm bu işlemleri gerçekleştirdikten sonra veritabanlarının oluşturulması için **"Form.Load"** metoduna **Context** sınıfından bir nesne ekleyerek **Create** metodunun çalışmasını sağlamanız yeterlidir. Projeyi çalıştırdığınızda boş bir form ekrana gelecek ve veritabanınız yaratılmış olacaktır.

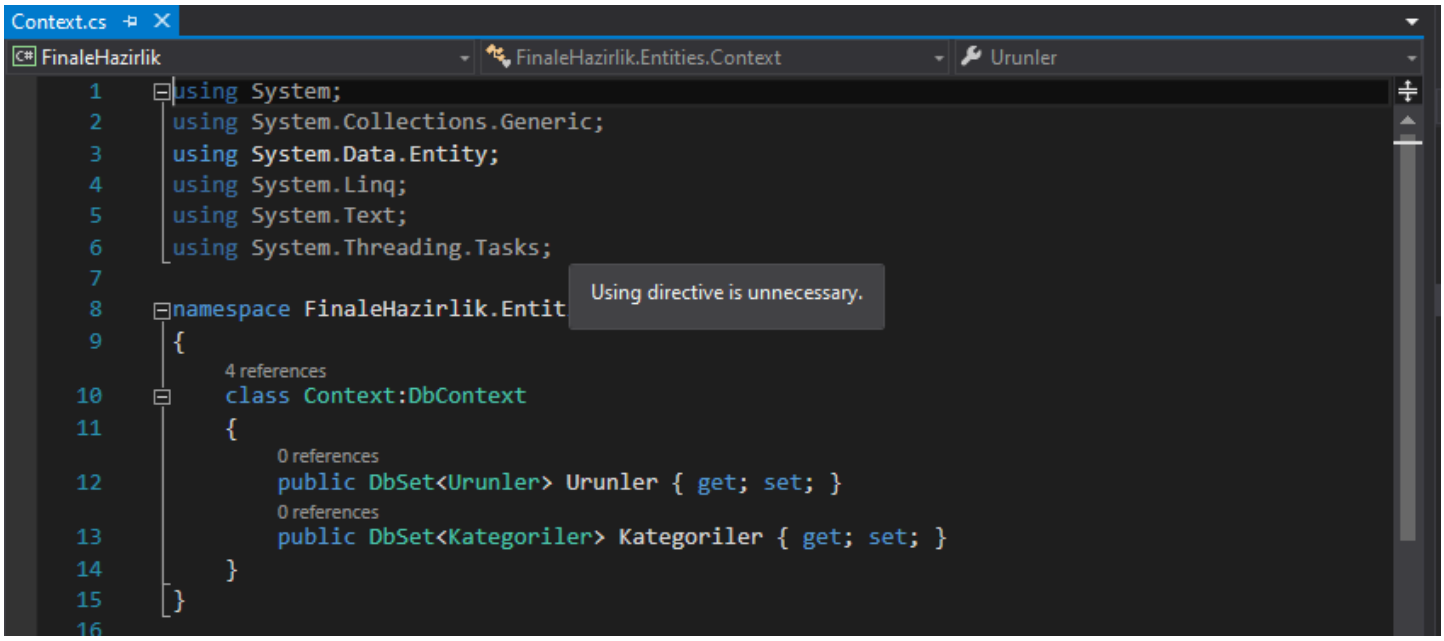
```
Form1.cs | Form1.cs [Design] | Context.cs | Urunler.cs
FinaleHazirlik | FinaleHazirlik.Form1 | Form1()
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10 using FinaleHazirlik.Entities;
11
12 namespace FinaleHazirlik
13 {
       3 references
14     public partial class Form1 : Form
15     {
           1 reference
16         public Form1()
17         {
18             InitializeComponent();
19         }
20
           1 reference
21         private void Form1_Load(object sender, EventArgs e)
22         {
23             Context baglanti = new Context();
24             baglanti.Database.Create();
25         }
26     }
27 }
```

Bu aşamadan sonra veritabanı üzerinde yapacağımız değişiklikleri migration kullanarak gerçekleştirebiliriz.

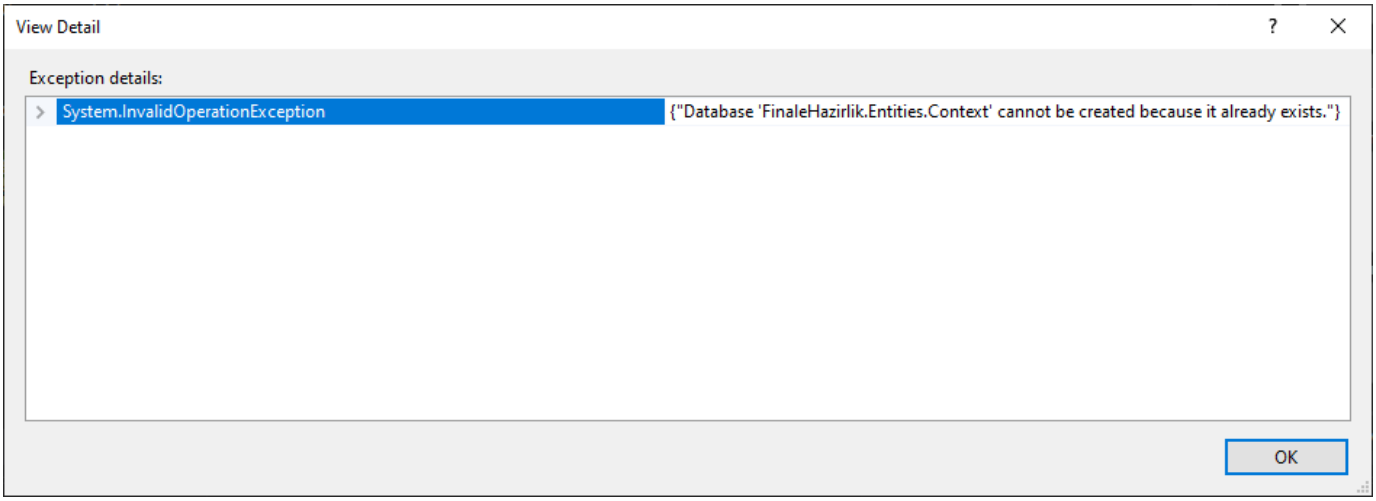
Örneğin; ürünleri kategorilere ayırmak istersek veritabanına bir kategoriler tablosu eklemek gereklidir. Bunun için; **"Solition Explorer"** panelindeki **"Entities"** klasörüne sağ tuşla tıklayarak açılan menüden sırasıyla **"Add" -> "Class"** yolunu takip ederek ürün kategori bilgilerini tutacak tabloyu oluşturmak için **"Kategoriler.cs"** sınıfını aşağıdaki özellikleri tanımlayarak projeye ekleyelim.

```
Kategoriler.cs | Form1.cs | Form1.cs [Design]
FinaleHazirlik | FinaleHazirlik.Entities.Kategoriler | KategoriID
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace FinaleHazirlik.Entities
9  {
       2 references
10     public class Kategoriler
11     {
12         [Key]
           0 references
13         public int KategoriID { get; set; }
           0 references
14         public string KategoriAdi { get; set; }
           0 references
15         public ICollection<Urunler> Urunler { get; set; }
16     }
17 }
```

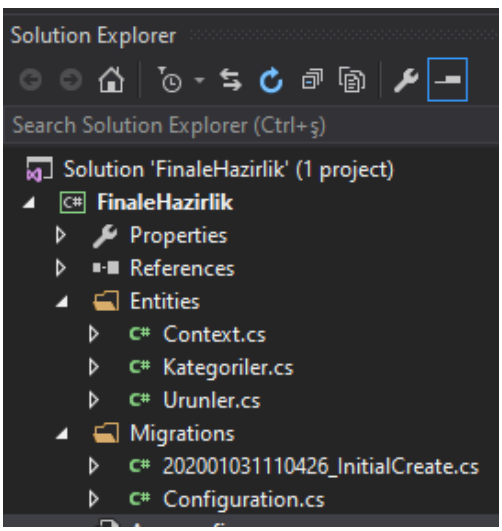
Context sınıfımızda da gerekli düzenlemeleri yapalım

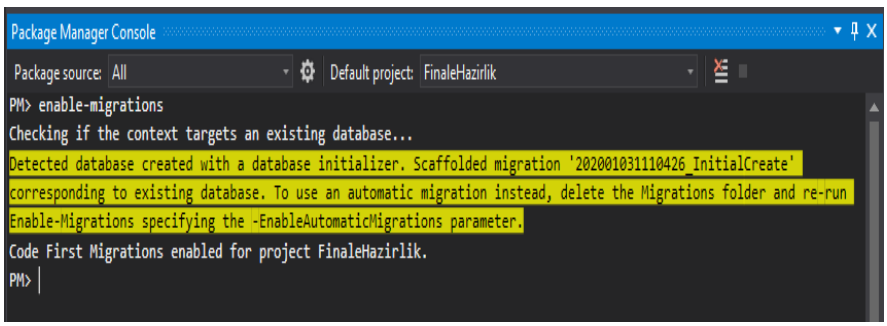


Projeyi tekrar çalıştırdığımızda; veritabanının yaratılamadığını, aynı isimde bir veritabanının zaten var olduğunu belirten bir hata mesajı alırız.

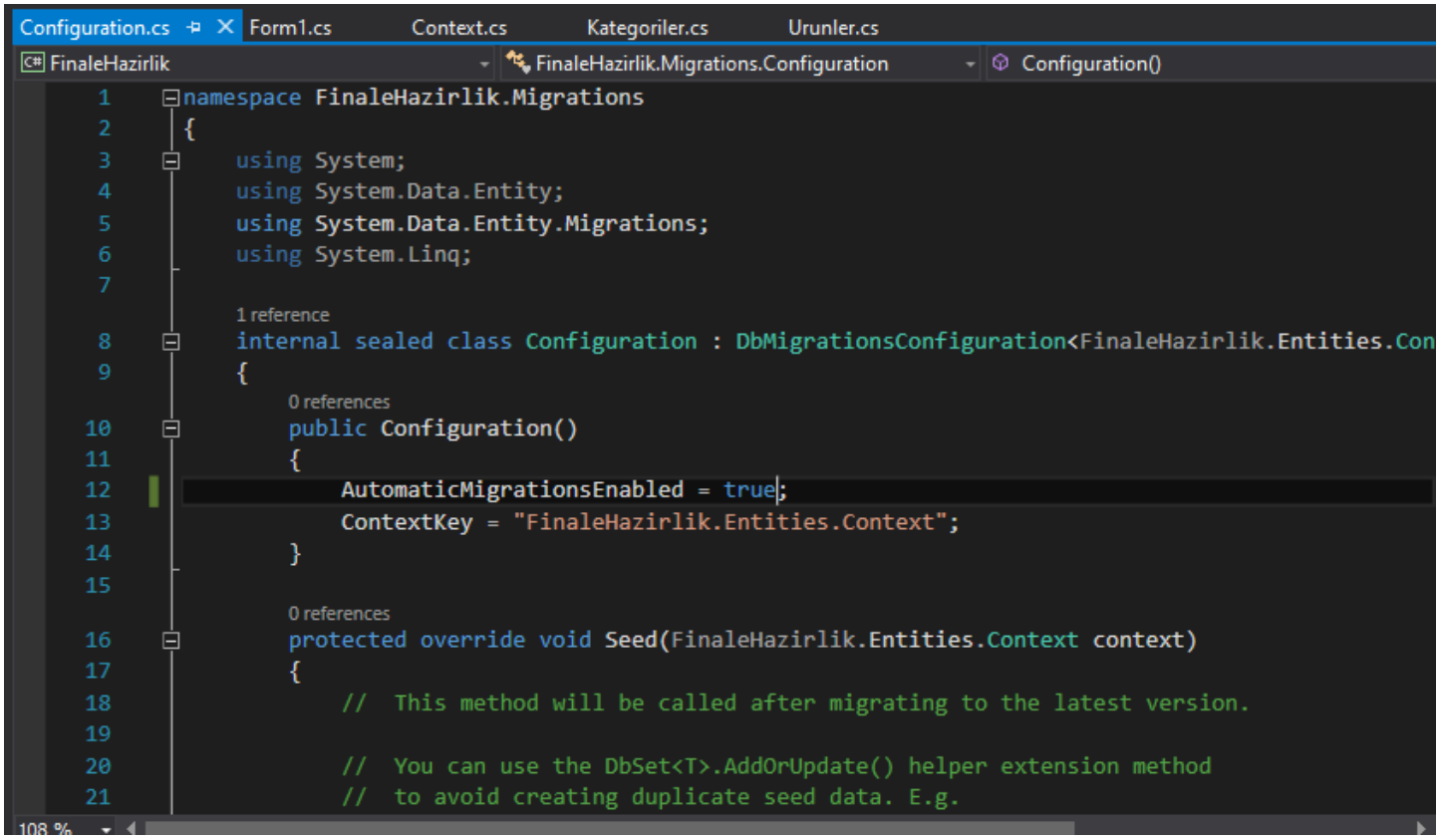


Bu sorunu aşmak için ya mevcut veritabanını silip yeniden oluşturmanız ya da **migrotion** kullanmanız gerekiyor. Migration yapısını aktifleştirmeniz için **"Package Manager Console"** penceresini kullanabilirsiniz. Konsol ekranında **"enable-migrations"** komutunu çalıştırdığınızda projenize **"Migrations"** adında bir klasör eklenecektir. Bu klasörde yer alan iki dosyadan **"Configurations"** isimli dosyada Migrations ayarları, diğer dosyada ise projede yapılan ve veritabanı üzerinde uygulanacak değişiklik bilgileri yer almaktadır.

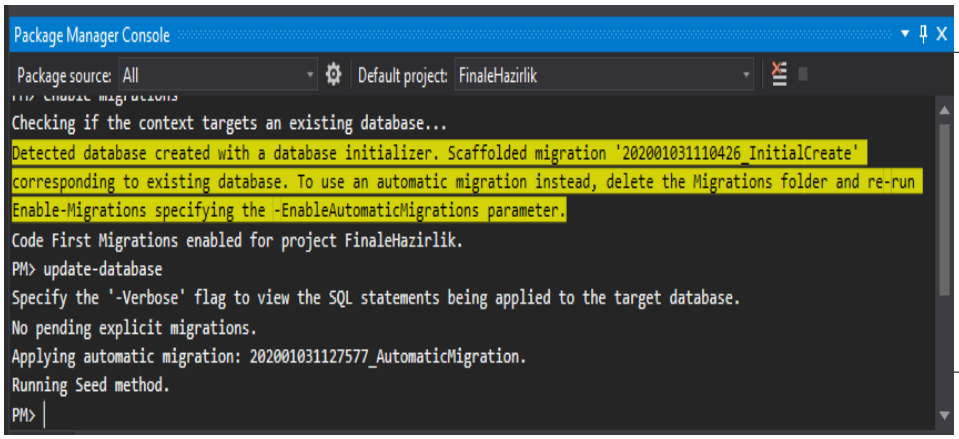


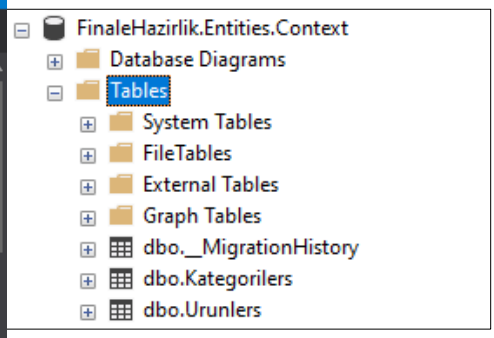


Oluşan konfigürasyon dosyasını incelediğinizde; “Configuration.cs” sınıfını “DbMigrationsConfiguration” sınıfından kalıtım alarak Context sınıfımıza işaretlenmiştir. Bunun yanında yapıcısı (constructur) nda **AutomaticMigrationsEnabled** özelliğinin varsayılan (default) olarak **false** olarak geldiğini görebilirsiniz. Migration işlemi için bu özeliği **true** olarak ayarlamanız gerekmektedir.

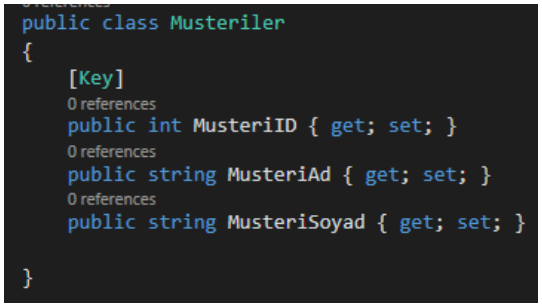


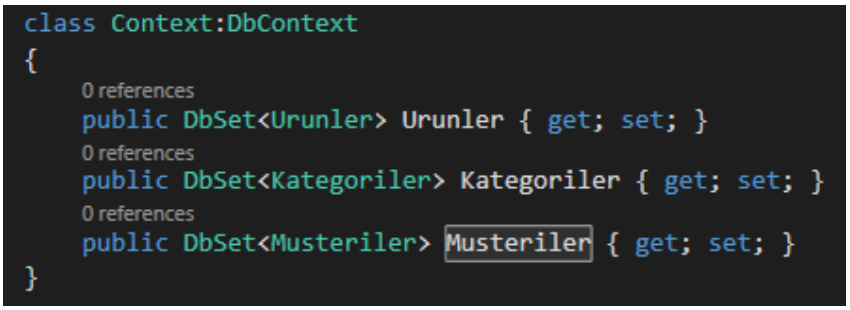
Migrations özelliğini aktif ettikten sonra yapılan değişikliklerin veritabanı üzerinde gerçekleşmesi için konsol ekranında **update-database** komutunu kullanmanız yeterlidir. Bu komutu kullanmanızla birlikte veritabanınızın güncellendiğini göreceksiniz.



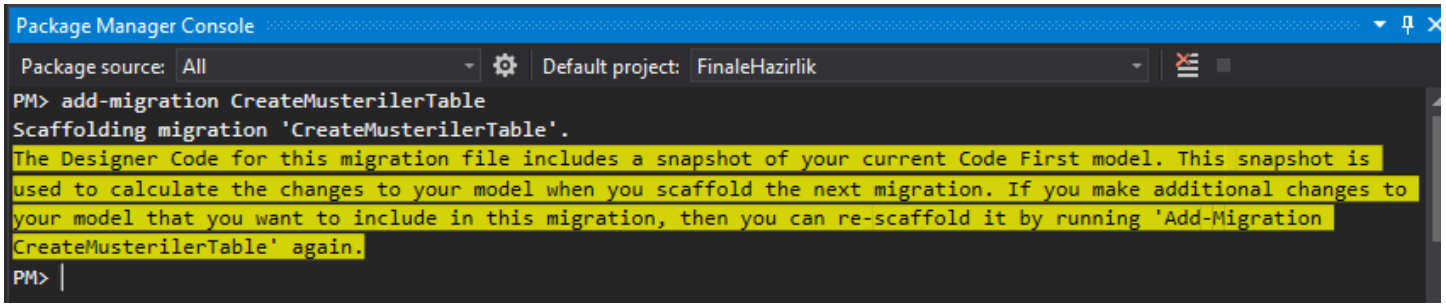


Şimdide müşterilerin bilgilerini tutacak **“Musteriler.cs”** sınıfını aşağıdaki özellikleri tanımlayarak projeye ekleyelim.

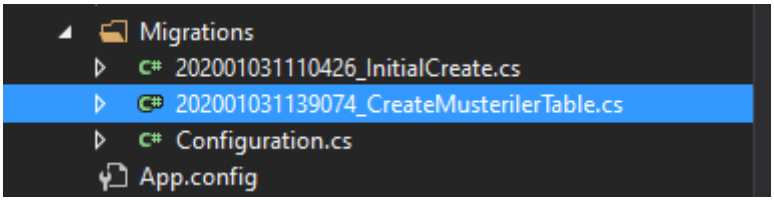




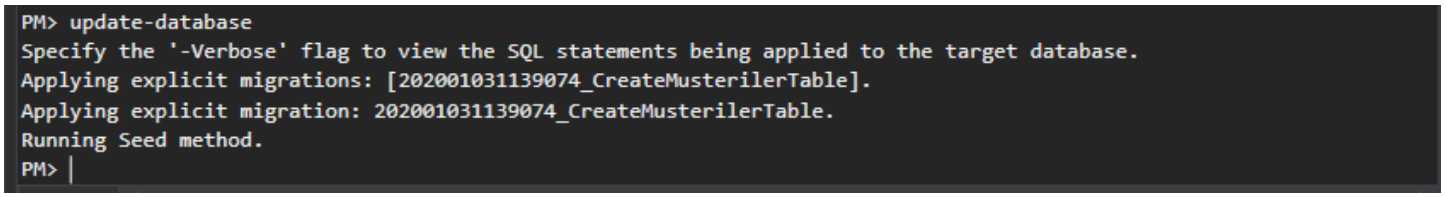
Context sınıfında da gerekli düzenlemeleri yaptıktan sonra **“Package Manager Console”** penceresini kullanarak yeni bir migration oluşturmanız gerekmektedir. Bunun için **“add-migration [migration ismi]”** komutunu kullanabilirsiniz. Migration ismi belirlerken **yapılan değişiklik ile ilgili bir isim vermek** migration sayısı arttığı zaman oldukça faydalı olacaktır.



Komutu uyguladıktan sonra projemizde aynı isimle bir migration oluşturulduğunu göreceksiniz.



Migration da hazırlandığına göre artık yapılan değişikliğin veritabanında da gerçekleşmesi için **“updata-database”** komutunu kullanabilirsiniz.



Benzer şekilde veritabanı üzerinde istediğiniz değişikliği (tablo ekleme, tablo silme, tabloda sütun ekleme – silme – isim değiştirme vb.) yapabilirsiniz. **Ekte indireceğiniz proje dosyasında benzer birkaç migration tanımlandığını görebilir ve migrationları inceleyebilirsiniz.**