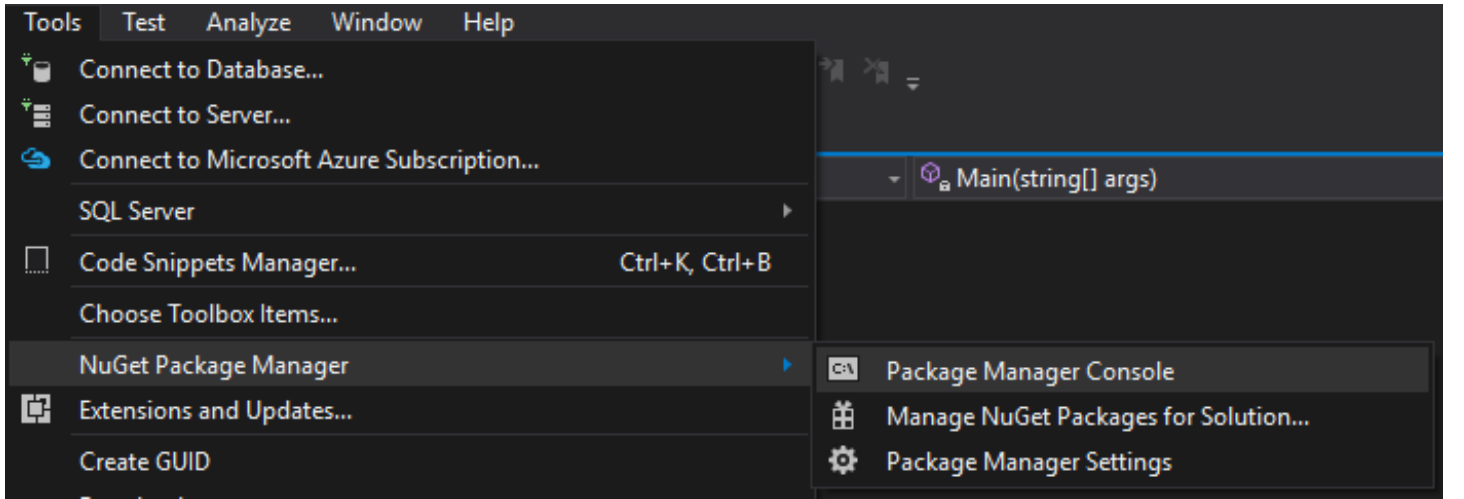


MIGRATIONS

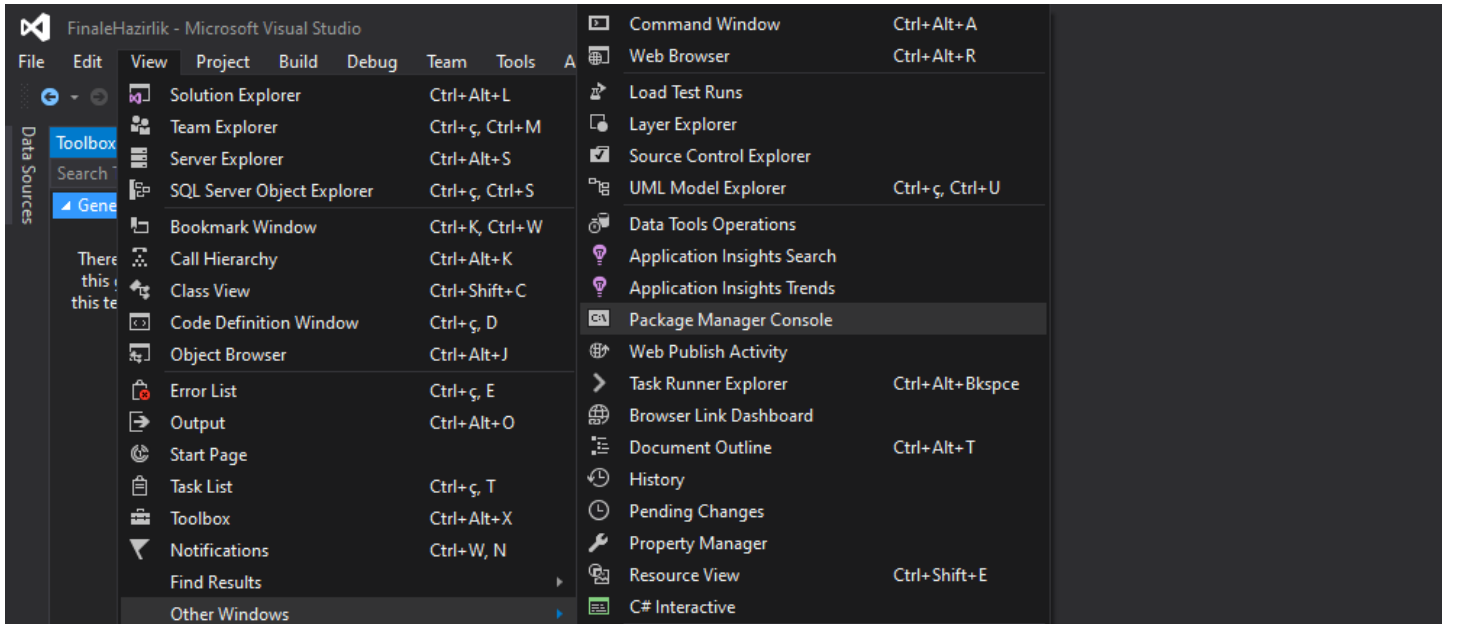
Code First deseni uygulanan bir projede veritabanına direkt olarak müdahale etmek oldukça sakıncalıdır. Haliyle yapacağımız ekleme, çıkarma veyahut güncelleme durumlarını tabloları temsil eden sınıflar (class) üzerinde gerçekleştirmeli ve güncellemeleri buradan gerçekleştirmeliyiz.

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.

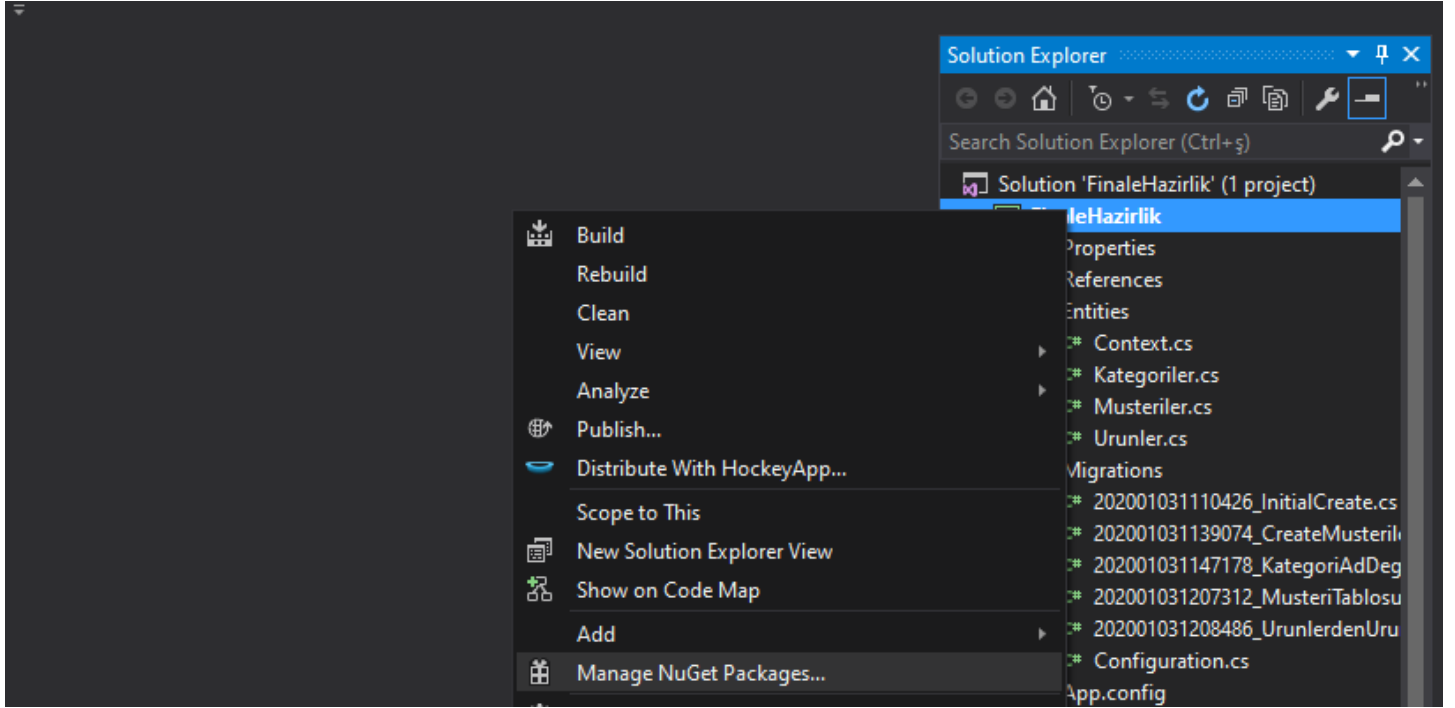


Visual Studio ile Code First deseni ile oluşturulan projelerde migration özelliği kapalı olarak gelir. Visual Studio üzerinden veritabanı üzerinde gerekli değişiklikleri yapabilmek için öncelikle bu özelliği aktif etmemiz gerekmektedir.

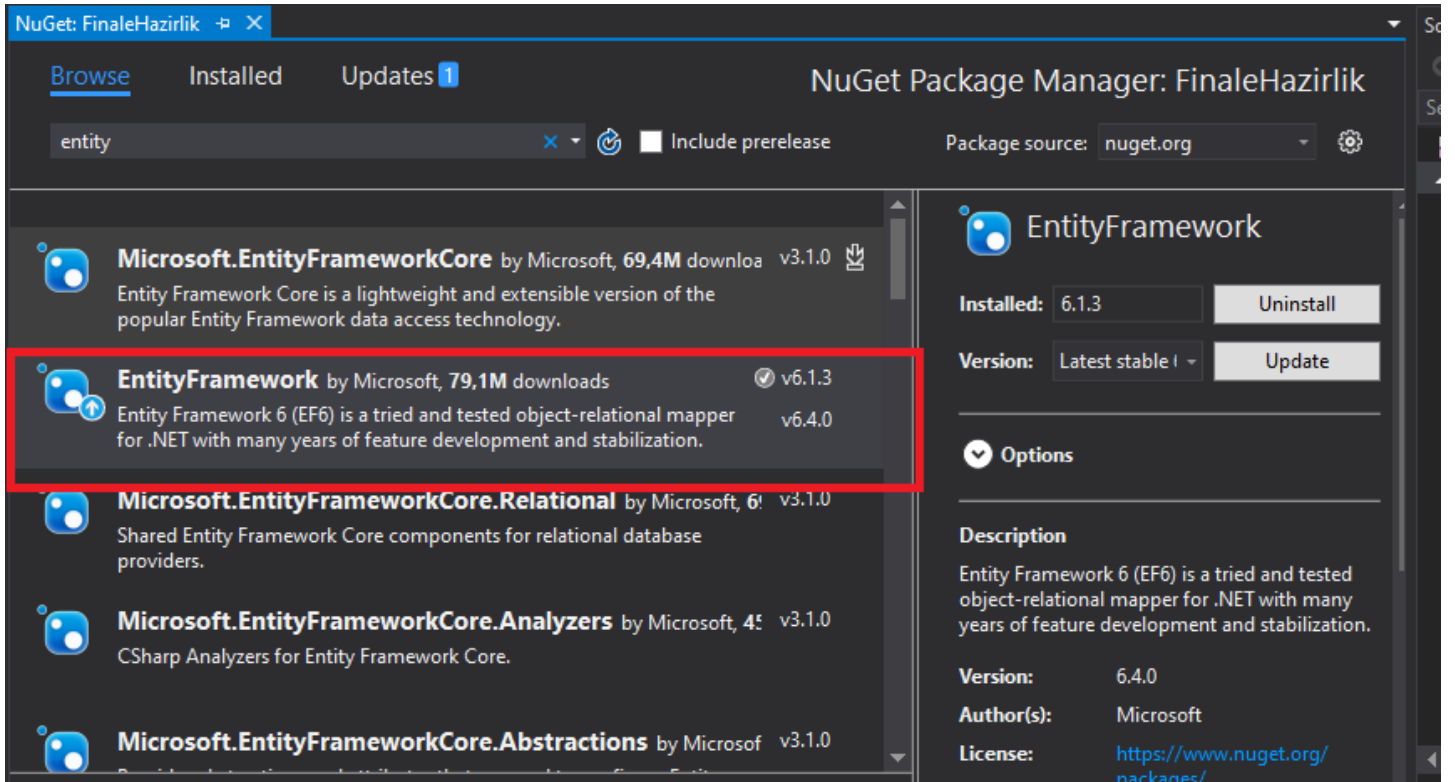
Basit bir proje üzerinde bu özellikleri gösterelim. Yapılacak işlemleri kısaca özetleyecek olursak

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.



3. "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.

```
Urunler.cs
FinaleHazirlik
FinaleHazirlik.Entities.Urunler
urunID

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using System.ComponentModel.DataAnnotations;
7
8 namespace FinaleHazirlik.Entities
9 {
10     2 references
11     public class Urunler
12     {
13         [Key]
14         0 references
15         public int urunID { get; set; }
16         0 references
17         public string urunAdi { get; set; }
18         0 references
19         public string urunMarka { get; set; }
20         0 references
21         public int urunStok { get; set; }
22         0 references
23         public Kategoriler Kategori { get; set; }
24     }
25 }
```

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.

```
Context.cs*
Urunler.cs
FinaleHazirlik
FinaleHazirlik.Entities.Context
Urunler

1 using System;
2 using System.Collections.Generic;
3 using System.Data.Entity;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace FinaleHazirlik.Entities
9 {
10     4 references
11     class Context:DbContext
12     {
13         0 references
14         public DbSet<Urunler> Urunler { get; set; }
15     }
16 }
```

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.

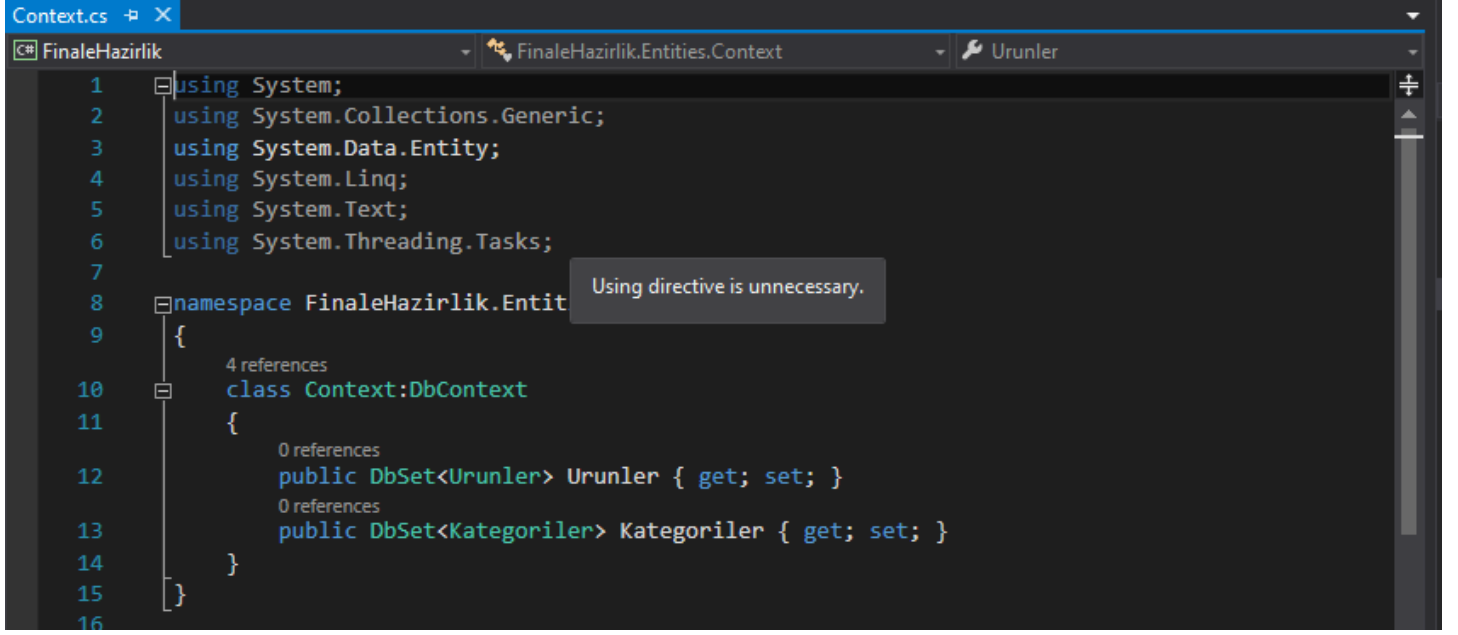
```
Form1.cs [Design] Context.cs Urunler.cs
C# FinaleHazirlik
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Threading.Tasks;
9 using System.Windows.Forms;
10 using FinaleHazirlik.Entities;
11
12 namespace FinaleHazirlik
13 {
14     public partial class Form1 : Form
15     {
16         public Form1()
17         {
18             InitializeComponent();
19         }
20
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.

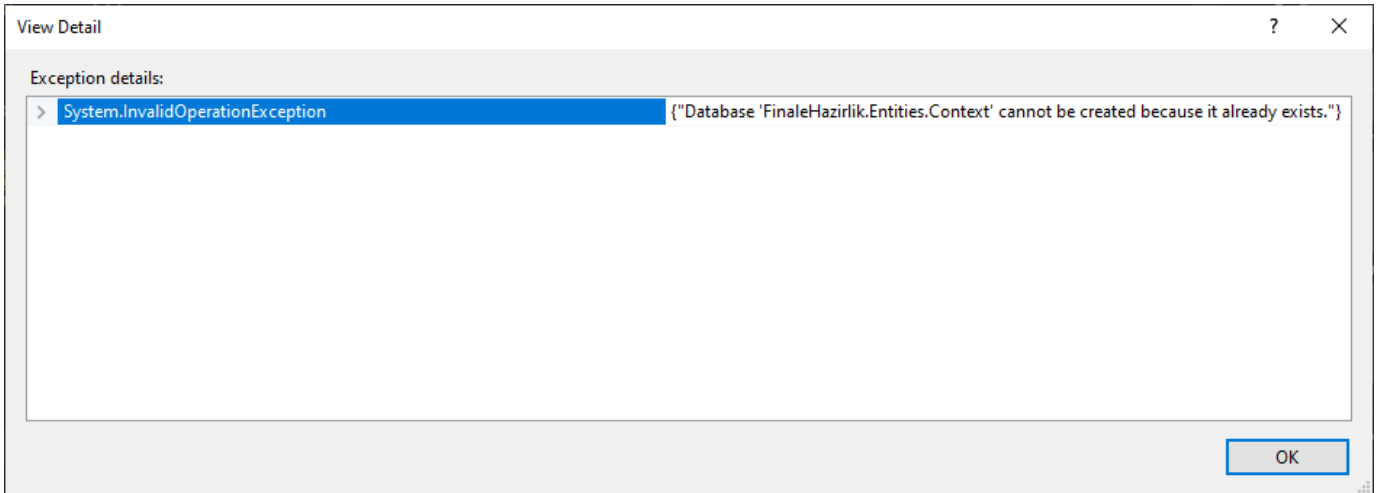
```
Kategoriler.cs Form1.cs [Design]
C# FinaleHazirlik
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel.DataAnnotations;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace FinaleHazirlik.Entities
9 {
10     public class Kategoriler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

Context sınıfımızda da gerekli düzenlemeleri yapalım

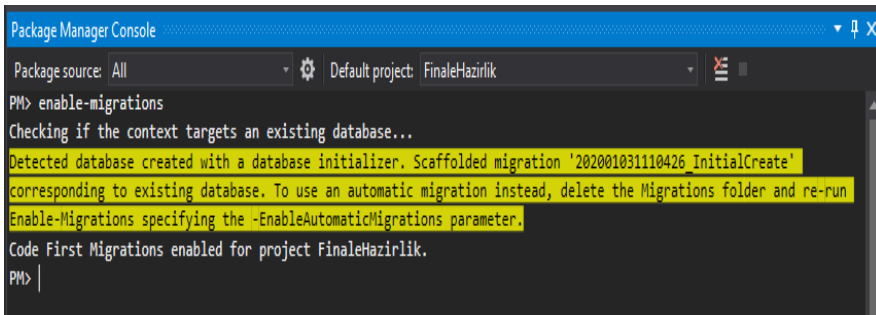


```
1 using System;
2 using System.Collections.Generic;
3 using System.Data.Entity;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace FinaleHazirlik.Entities
9 {
10     class Context:DbContext
11     {
12         public DbSet<Urunler> Urunler { get; set; }
13         public DbSet<Kategoriler> Kategoriler { get; set; }
14     }
15 }
16
```

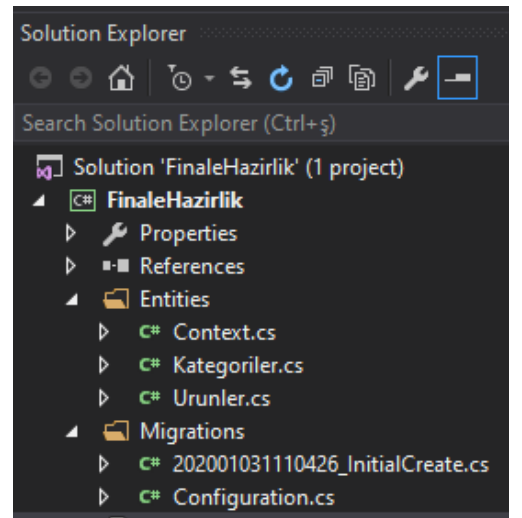
Projeyi tekrar çalıştırdığımızda; veritabanının yaratılmadığını, aynı isimde bir veritabanının zaten var olduğunu belirten bir hata mesajı alırız.



Bu sorunu aşmak için ya mevcut veritabanını silip yeniden oluşturmanız ya da **migration** 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.



```
Package Manager Console
Package source: All Default project: FinaleHazirlik
PM> enable-migrations
Checking if the context targets an existing database...
Detected database created with a database initializer. Scaffolded migration '202001031110426_InitialCreate'
corresponding to existing database. To use an automatic migration instead, delete the Migrations folder and re-run
Enable-Migrations specifying the -EnableAutomaticMigrations parameter.
Code First Migrations enabled for project FinaleHazirlik.
PM>
```



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

```
Configuration.cs Form1.cs Context.cs Kategoriler.cs Urunler.cs
FinaleHazirlik FinaleHazirlik.Migrations.Configuration Configuration()

1 namespace FinaleHazirlik.Migrations
2 {
3     using System;
4     using System.Data.Entity;
5     using System.Data.Entity.Migrations;
6     using System.Linq;
7
8     1 reference
9     internal sealed class Configuration : DbMigrationsConfiguration<FinaleHazirlik.Entities.Con
10    {
11        0 references
12        public Configuration()
13        {
14            AutomaticMigrationsEnabled = true;
15            ContextKey = "FinaleHazirlik.Entities.Context";
16        }
17
18        0 references
19        protected override void Seed(FinaleHazirlik.Entities.Context context)
20        {
21            // This method will be called after migrating to the latest version.
22
23            // You can use the DbSet<T>.AddOrUpdate() helper extension method
24            // to avoid creating duplicate seed data. E.g.
```

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.

```
Package Manager Console
Package source: All Default project: FinaleHazirlik
Checking if the context targets an existing database...
Detected database created with a database initializer. Scaffolded migration '202001031110426_InitialCreate'
corresponding to existing database. To use an automatic migration instead, delete the Migrations folder and re-run
Enable-Migrations specifying the -EnableAutomaticMigrations parameter.
Code First Migrations enabled for project FinaleHazirlik.
PM> update-database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
No pending explicit migrations.
Applying automatic migration: 202001031127577_AutomaticMigration.
Running Seed method.
PM>
```

```
FinaleHazirlik.Entities.Context
+ Database Diagrams
+ Tables
+ System Tables
+ FileTables
+ External Tables
+ Graph Tables
+ dbo._MigrationHistory
+ dbo.Kategorilers
+ dbo.Urunlers
```

Şimdi müşterilerin bilgilerini tutacak "Musteriler.cs" sınıfını aşağıdaki özellikleri tanımlayarak projeye ekleyelim.

```
public class Musteriler
{
    [Key]
    0 references
    public int MusteriID { get; set; }
    0 references
    public string MusteriAd { get; set; }
    0 references
    public string MusteriSoyad { get; set; }
}
```

```
class Context:DbContext
{
    0 references
    public DbSet<Urunler> Urunler { get; set; }
    0 references
    public DbSet<Kategoriler> Kategoriler { get; set; }
    0 references
    public DbSet<Musteriler> Musteriler { get; set; }
}
```

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.

```
Package Manager Console
Package source: All Default project: FinaleHazirlik
PM> add-migration CreateMusterilerTable
Scaffolding migration 'CreateMusterilerTable'.
The Designer Code for this migration file includes a snapshot of your current Code First model. This snapshot is used to calculate the changes to your model when you scaffold the next migration. If you make additional changes to your model that you want to include in this migration, then you can re-scaffold it by running 'Add-Migration CreateMusterilerTable' again.
PM> |
```

Komutu uyguladıktan sonra projemizde aynı isimle bir migration oluşturulduğunu göreceksiniz.

```
└─ Migrations
  └─ C# 202001031110426_InitialCreate.cs
  └─ C# 202001031139074_CreateMusterilerTable.cs
  └─ C# Configuration.cs
  └─ App.config
```

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

```
PM> update-database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [202001031139074_CreateMusterilerTable].
Applying explicit migration: 202001031139074_CreateMusterilerTable.
Running Seed method.
PM> |
```

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.**