|  |  |
| --- | --- |
| EnTİTY fRAMEWOK  CORE | Berkay Akar |

İçindekiler

[ENTİTY FRAMEWORK VE ORM NEDİR ? 2](#_Toc98869019)

[KURULUM 3](#_Toc98869020)

[Context Sınıfının Oluşturulması 7](#_Toc98869021)

[Veritabanını Oluşturma Ve ilk Migration İşlemi 7](#_Toc98869022)

[Veritabanı İşlemleri 8](#_Toc98869023)

[Veritabanına Veri Ekleme : 8](#_Toc98869024)

[Veritabanindan Kayıt Getirme 10](#_Toc98869025)

[Kayıt Güncelleme 12](#_Toc98869026)

[Veritabanından Bir Kayıt Silme 15](#_Toc98869027)

[Provider Değişikliği İle Tüm Projeyi Çeşitli Veritabanları İçin Uygun Hale Getirmek 16](#_Toc98869028)

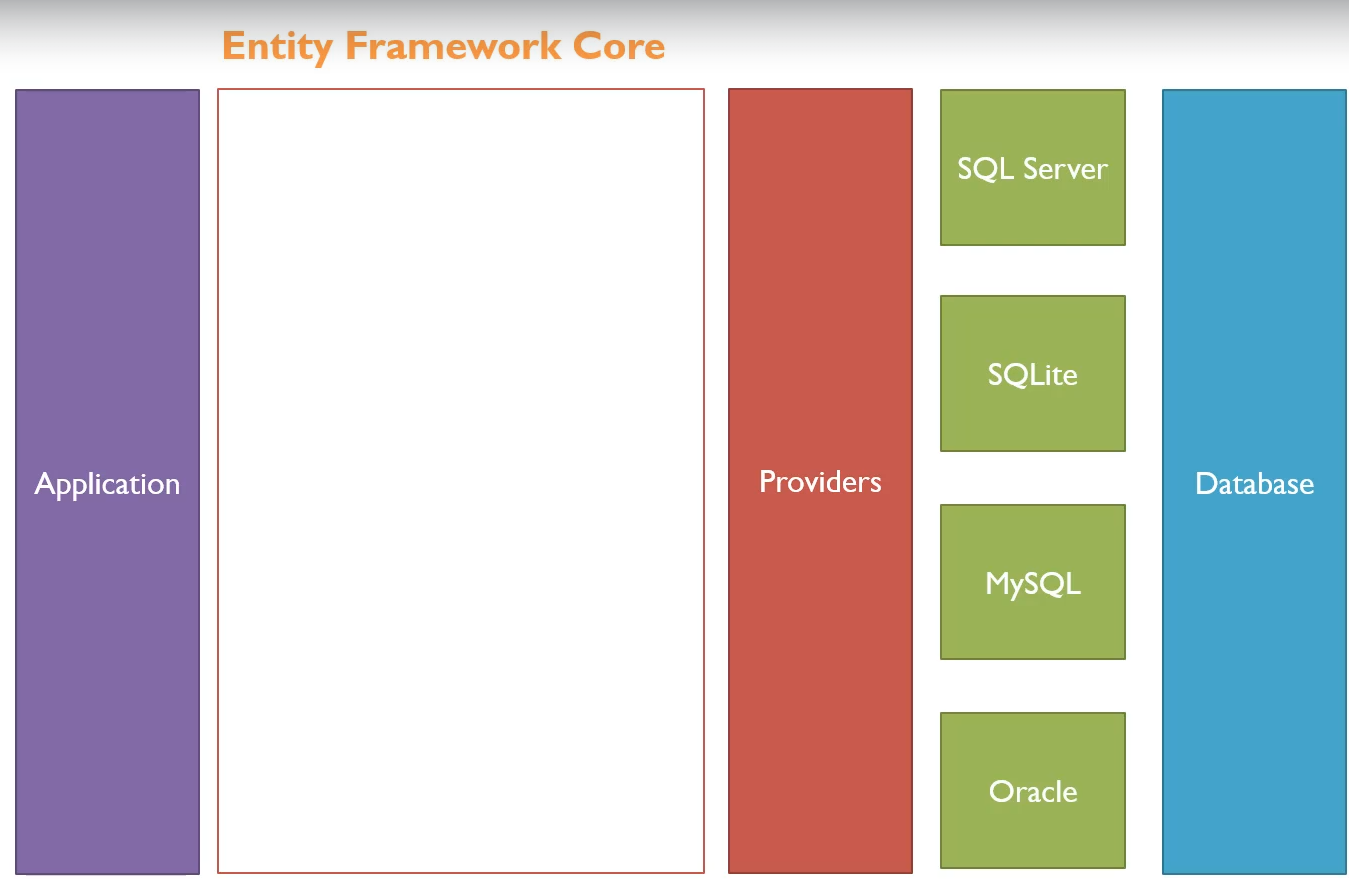
[ENTİTY FRAMEWORK CORE İLİŞKİLER 17](#_Toc98869029)

[One To Many (1-N) 17](#_Toc98869030)

# ENTİTY FRAMEWORK VE ORM NEDİR ?

Bir uygulamada veritabanına erişim kurarken veri tabanına erişim için bir Provider (driver) kurmamız gerekmektedir. ADO.NET teknolojilerinden de hatırlanabileceği gibi ilk olarak bir connection nesnesi, Beraberinde bir command nesnesi ve özellikleri verilerek bu işlem gerçekleştiriliyordu. Bu işlem için bir sql sorgusu yazmamız gerekmekteydi. Bağlantıyı isteğe bağlı olarak Connected ve disconnected olarak tanımladıktan sonra model dosyaları doldurularak uygulama içerisinde kullanılabilmekteydi. Bu işlem Mapping olarak adlandırılmaktadır. Mappingten kastımız sql içerisinden dönen verinin uygulamada kullanılabilir bir veri tipine göre eşleştirilerek kullanılması sürecidir.

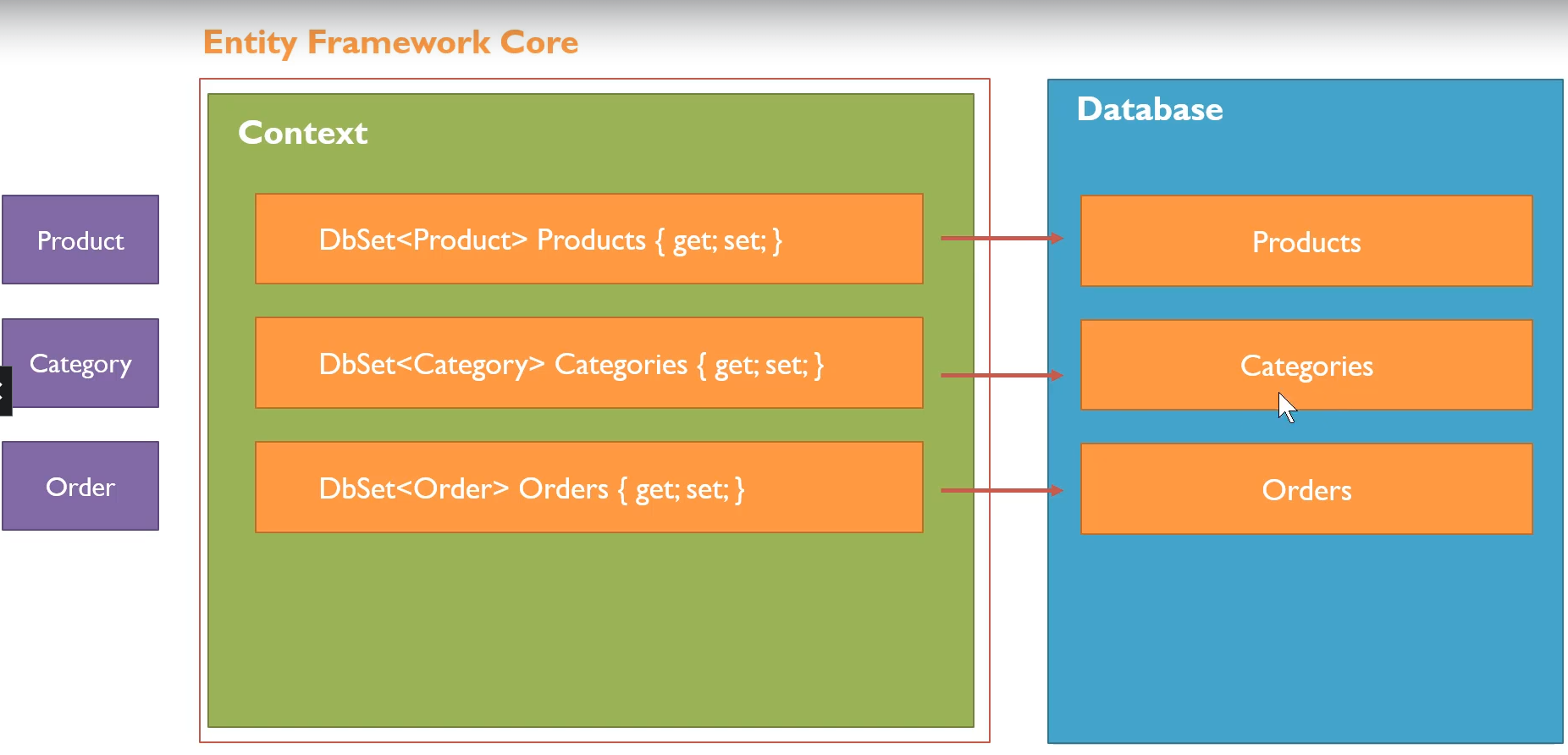
Bu süreci daha kolay olarak gerçekleştiren araçlar üretilmiştir. Bu araçlar Object Relaiton Mapping (**ORM**) olarak adlandırılmaktadır. ORM Birçok Veritabanı Yönetim Sistemini desteklemektedir.



Uygulama ile Entity Framewok ilerişime geçebilmesi için

1. Entity Framework Core
2. Hangi Veri Tabanı Yönetim Sistemi İle Çalışılacaksa onun bir Provider I

Yüklenmesi gerekmektedir.



# KURULUM

Yukarıda işleyiş bahsedildiği üzere hedef veri tabanı belirlenerek bu veritabanına uygun provider kurulumu gerçekleştirilmeldir. Microsoft sitesi veya Nuget Package Manager bu işi sağlayabilen servislerdir.

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 1 Providerlar üzerinden örnek

Örnek oluşturulabilmesi açısından işleyişe SQLİTE database I için gerekli provider tedariği ile başlıyoruz. ORM yapısındaki kodlama merkezi bir yönetim oluşturduğu için sadece provider değişiminde tüm db lere dönüşümü sağlanabilmektedir.

Bir console uygulaması açalım

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Uygulama 5.0 da oluşturulduğundan ötürü paketleri yüklerken uyumluluklarını göz önüne alarak indiriniz.

metin, ekran görüntüsü, ekran içeren bir resim

Açıklama otomatik olarak oluşturuldu

Paket kurulumunu lütfen ilgili projenin dizinine gerçekleştiriniz. Solution üzerine paket kurulması durumunda tüm proje boyunca paketler yüklenmiş olacak ve bu projenin hatalı üretim (deising patternlerde uyumsuzluk ve güvensizlik ), fazla yer kaplaması gibi çeşitli problemleri doğruacaktır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Core yazmayan (en üstteki) .NetFramework için gerekli pakettir.

metin, ekran görüntüsü, ekran, iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu

Provider da budur.

Kurulacak bir diğer kütüphane ise db-first, migration komutlarının içinde yer aldıgı EntityFrameworkCore.Tools kütüphanesidir.

metin, ekran görüntüsü, ekran, siyah içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 2 Tools Kütüphanesi ve içeriğindeki kodlar

Bir sonraki aşamada Veritabanı tablolarını kodlamaya başlamak için bir adet Entities isimli bir klasör oluşturup içerisine tablolarımızı DataAnnodation komutları ile işlemeye başlayacağız.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Entitiler burada sınıflar şeklinde oluşturularak ve üzerinden migration denen bir yöntem ile veritabanı tablolarına dönüştürülecektir.

Tablo oluştururken dikkat edilmesi gereken özellik ise her tablonun bir primary key özelliğinin bulunması zorunluluğudur.

public int ID { get; set; }

public decimal Price { get; set; }

Alanları bu şekilde belirlenmiş olan bir sınıfta eğer alan isimlerinden birisi sadece Id veya ClassAdıId şeklinde tanımlanırsa hiçbir data annodation kullanmadan veritabanı bunu primary key olarak algılar. Fakat farklı bir isim koyarsak üzerine gelerek [Key] attributunu eklememiz gerekecektir.

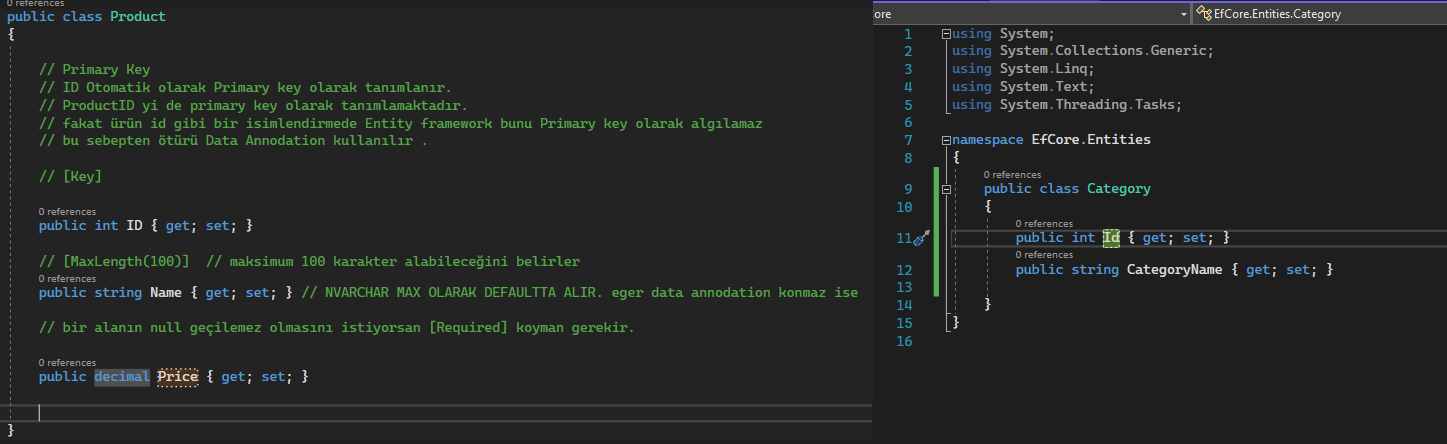


Figure 3 Entitiler Bu şekilde tanımlandı

# Context Sınıfının Oluşturulması

Bu aşamada bir adet Context isimli sınıf açılır. Bu sınıfın bir Context olduğunun algılanması için DbContext sınıfından bir kalıtım verdirtilmesi gerekmektedir. Eklenen verilerin Çalıştırılabilmesi için tabloların tek tek Dbset<sınıf> formatında bir tanımlama ile birlikte bulunulması gerekmektedir. Bu property lere verilecek her isim Veritabanı kısmında tablo ismine karşılık gelineceğini unutmayalım. Beraberinde bir diğer husus veritabanını nerede oluşturulacağıdır. İsteğe bağlı olarak ConnectionString tagı ile proje üzerinde , Appconfigte veya DbContext ten kalıtım alan yerde OnConfigüring komutu ezdirilerek connection stringi tanınabilir. OptiunBuilder den sonra UseHangidbtipi() fonksiyonu ile (Provider olarak kurulan kütüphane hangisi ise onun methodu burada belirlenir. ) bu şekilde işe başlanır .

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

# Veritabanını Oluşturma Ve ilk Migration İşlemi

Veritabanı kısmında (Code kısımıda fiziksel veritabanı değil) her bir değişiklikte değişikliğin veritabanına işlenebilmesi için Migraton oluşturma ve bu migration ile veritabanına update ederek kullanmamız gerekmektedir.

Son aşamada 4 adet paket kurulu olması gerekmektedir.

* EntityFrameworkCore (ef için)
* EntityFrameworkCore.Sqlite (hedef provider için)
* EntityFramewok.Tools (update,migration gibi yapılar için )
* EntityframeworkCore.Deisinger (Dbset ve Context sınıfı için)

Bu aşamadan sonra yapılması gereken add-migration migrationismi , update-database ile veri tabanını update ederek kullanılabilir bir db oluşturmaktadır.

# Veritabanı İşlemleri

## Veritabanına Veri Ekleme :

using (var context = new SQLİTECONTEXT())

{

var p = new Product { Name = isim , Price = fiyat };

context.Products.Add(p);

context.SaveChanges();

Console.WriteLine("İşlem başarılı");

ProductGetir();

}

SQLİTECONTEXT sınıfı içerisinde kalıtım yolu ile DbContext sınıfından Dispose method gelidiği için var değişkeni ile kullanılabilmektedir. Temel ekleme, güncelleme, silme ve görüntüleme işlemlerinde kendi Context sınıfımızdan üretilmiş bir context . bunun bir tabosu . method ismi ve parametresi verilmelidir. İşlem bittikten sonra ise context.saveChanges ile verilen işi veritabanına comitleyerek eşleyebilmekteyiz.

Bir context e birden fazla veri aynı anda atılmak istenirse listelerden de tanıgımız metho d olan addrange kullanılabilmektedir .

using (var context = new SQLİTECONTEXT())

{

List<Product> productlistesi = new List<Product>() {

new Product { Name = "samsung s6", Price = 2500 },

new Product { Name = "samsung s7", Price = 3500 },

new Product { Name = "samsung s8", Price = 4500 },

};

context.Products.AddRange(productlistesi);

context.SaveChanges();

Console.WriteLine("İşlem başarılı");

Yürütülen sorguların SQL karşılıklarını ve en son çalıştırılan sorgunun sql karşılığını entity frameworkte alabilmemiz mümkündür. Bu işlemin türkçesine loglama Entity Frameworkte ki kaşılığına logging denmektedir. Bu iş için Eklenmesi gereken bir kütüphanemiz mevcuttur.

metin, ekran görüntüsü, ekran, siyah içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bir sonraki aşamada ise DbContext sınıfımızdan kalıtım almış olan SQLİTECONTEXT sınıfı içerisinde eklenmesi gereken bir ILogger ekleyelim .

public static readonly ILoggerFactory loggerFactory = LoggerFactory.Create(builder => { builder.AddConsole(); });

Override ettiğimiz onconfiguring kısmındaki Provider ın önüne bir değişken daha eklenerek sonuç bitirilir.

Son hali aşağıdaki şekilde verilmiştir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu işlemden sonra yapılabilecek her türlü insert update delete işleminde bize hedef makinede nasıl bir sql çalıştırdığını gösterecektir. Soldaki kodun çıktısı sağ kısımda verilmiştir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

## Veritabanindan Kayıt Getirme

Bir context Referansı oluşturularak bu context üzerinden tablomuzu vermeil ve üzerinden bir LİNQ çalıştırarak tablolarımıza erişebilmekteyiz. Kısacası :

using (var context = new SQLİTECONTEXT())

{

var tablo = context.Products.ToList();

Console.WriteLine("ÜRÜNLER TABLOSU :");

foreach (var item in tablo)

{

Console.WriteLine("ID : "+item.ID + " İSİM :"+item.Name+" FİYAT : "+item.Price);

}

}

Kodun çıktısı Aşağıda verilmiştir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tolist methodu ile veri tabanından tüm veriler dönemktedir. Bizlere lazım olan verileri (kolonları) **Select()** linq ile seçebilmekteyiz.

metin, ekran görüntüsü, ekran, gümüş içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görüldüğü üzere tolist yapmadan önce araya bir select koyarak bu şekilde kolonlarımızı seçebilmekteyiz.

Yukarıdaki filitrelemede sadece ID ve name bilgisi geldiğinden ötürü aşağıda price kolonu çıkmamıştır. Kodun çıktısı:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Sadece Tek bir kolon çekilmek isteniyor ise select (x=> x.kolon adı yazılabilir ). Anynomus tip e gerek yoktur

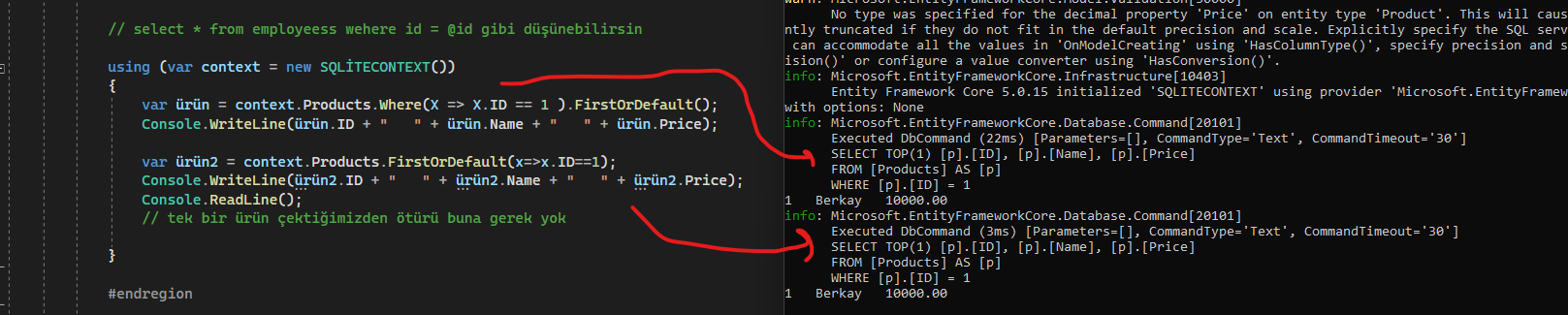
metin, bilgisayar, iç mekan, dizüstü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görüldüğü üzere sadece fiyat bilgisi sorgu olarak çekildi.

**WHERE:**

Belirli Bir koşula göre ürün çekmek istersek bu aşamada where keyword ünü kullanabilmektyeyiz. Kod karşılığı bir alt satırda mevcuttur.



Bu işlemde context.product tan sonra where koşulumuzu belirledikten sonra firstordefault atarak gelen bir verinin sadece bir nesne olduğunu, bunuda model olarak aktarmamız (tek bir veri olduğu için listeye gerek yok) gerektiğini belirterek bu şekilde bi atama gerçekleştiridik. Istersek koşulu where de istersek firstordefault içerisinde de belirtebiliriz.

First kullanılsaydı db den boş veri gelme durumunda program patlatacaktı. Bu sebeple firstordefault methodunun kullanılması daha yerinde bir süreçtir. Istersek biz bu süreçte en sonra tolist te kullanabilirdik fakat tek elemanlı bir listeyi foreach ile dönmemiz gerekeceğinden ötürü bu kullanım daha uygun olacaktır.

Bir örnek yapmak gerekirse fiyatı 500 ile 3500 arasında olan ürünleri listeleyelim.

using (var context = new SQLİTECONTEXT())

{

var tablo = context.Products.

Select(p => new { KİMLİK\_NO = p.ID, URUN\_ADİ = p.Name, FİYAT = p.Price }) .Where(f => f.FİYAT > 500 && f.FİYAT<4000 );

Console.WriteLine(" 500 ile 4000 arasında olan ürünler :");

foreach (var item in tablo)

{

Console.WriteLine(item.URUN\_ADİ + " " + item.KİMLİK\_NO + " " + item.FİYAT);

}

}

**Örnek :** Dışarıdan girilen isme göre ürün filitreleme sorgusunu yazınız

void getbylikename(string isim)

{

using (var context = new SQLİTECONTEXT())

{

var tablo = **context.Products.Where(x => x.Name.Contains(isim)).ToList();**

Console.WriteLine(isim + " İsmine ait toplam :" +tablo.Count + " adet veri listelenmiştir " );

foreach (var item in tablo)

{

Console.WriteLine(item.ID + " " + item.Name + " " + item.Price );

}

}

}

## Kayıt Güncelleme

İlk aşamada güncelleme işlemi için veritabanından bir veri alınmalıdır. Güncelleme ekranında (Product Detail Page) ürünün model olarak db den geldiğini düşünelim. Burada AKAR isimli ürünü tablomuzdan getiriyoruz. Akar isimli ürünümüzün id Bilgisi : 5 ;

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ürüne Select Atarak db den uygulamamıza getiriyoruz.

var table = context.Products.Where(x => x.ID == 5).FirstOrDefault();

ürün bir tane geleceği için sonuna first or default ekledik.

Gelen ürünün istenilen bilgilerini değiştirelim

table.Price = Convert.ToInt32(Console.ReadLine());

table.Name= Console.ReadLine();

Ürün eklemede de olduğu gibi bu aşamadan sonra ürünü context e yollarken:

Context.Products.Update(table);

Değişiklikleri commit edebilmek (veri tabanına işleyebilmek için)

Context.SaveChanges();

Süreçten anlaşılacağı gibi bir veri üzerinde işlem yapabilmek için ilk olarak db den çağırılması gerekmetkedir. EntityFramework bu süreci change tracking olarak değerlendirir. Değerlendirmede veri tabanından dönen veri süreç içerisinde bir nesne olarak tutulduğu ve üzerindeki değişiklikleri otomatikl olarak algıladığı için bu işlem mümkjündür. Siz sisteme id si 5 olan elle oluşturulmuş bir Product nesnesini update atmaya kalkarsanız change Tracking sürecini göz ardı ettiğiniz için system sizlere hata verecektir. Bir aşağıdaki kod buna örnektir. Ama siz elle oluşturulmuş olan bir kaydın change tracking ile eklemek gerekiyorsa attached methodunu kullanmamız gerekecektir.

using (var context = new SQLİTECONTEXT())

{

var table = context.Products.Where(x => x.ID == 5).FirstOrDefault();

if (table!= null)

{

Console.WriteLine(" 5 numaralı ürününüz bulundu ");

Console.WriteLine(table.ID + " " + table.Name + " " + table.Price);

Console.WriteLine();

Console.WriteLine("Değiştirmek istediğiniz bilgieri giriniz : ");

Console.WriteLine("fiyat : ");

table.Price = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("İsim : ");

table.Name= Console.ReadLine();

context.Products.Update(table);

context.SaveChanges();

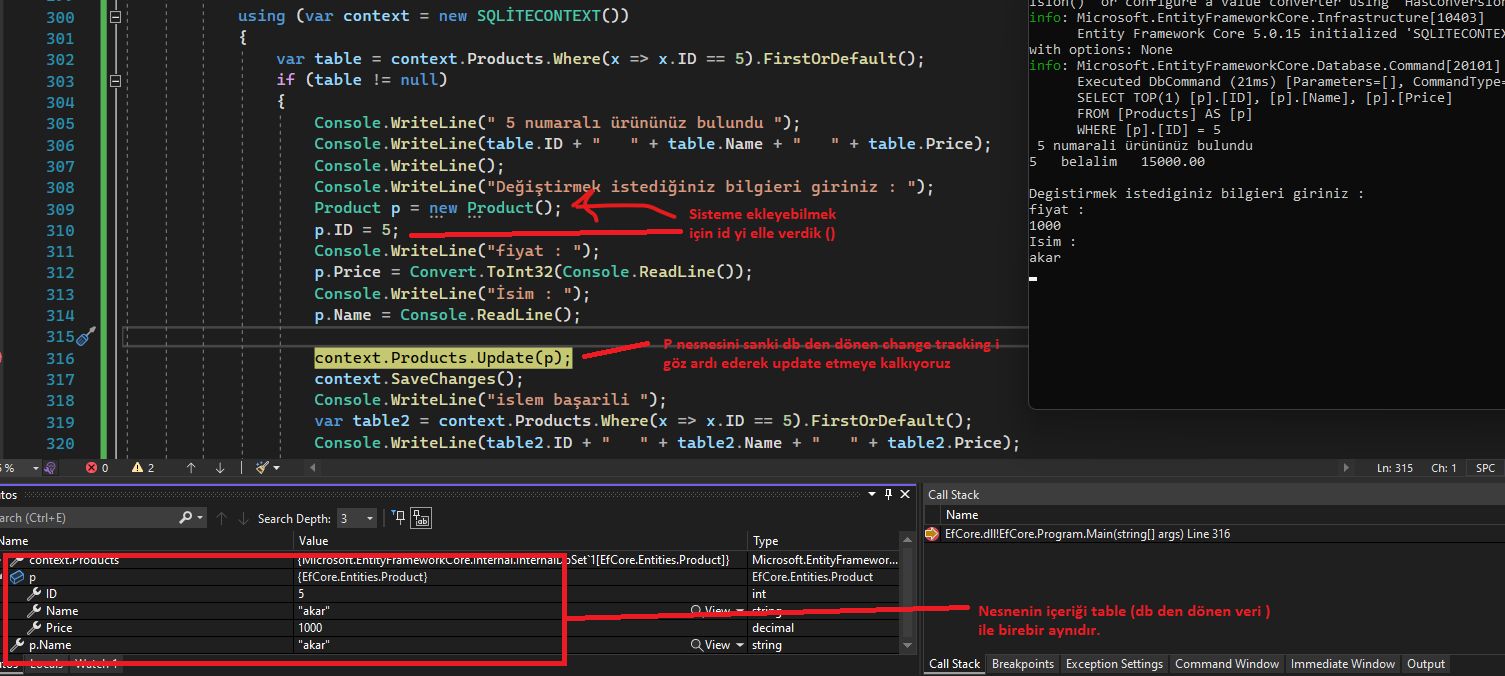
Console.WriteLine("islem başarili ");

var table2 = context.Products.Where(x => x.ID == 5).FirstOrDefault();

Console.WriteLine(table2.ID + " " + table2.Name + " " + table2.Price);

}

}



Bir sonraki aşamada kodu bir satır daha ilerletirsek hata iler karşılaşmaktayız.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu işlemi gerçekleştirebilmemiz için yeni oluşturulan nesneyi tracking methodu (Nesne takibi yöntemimin adı tracking ama veritabanında context e bağlayacağımız method ismi **Attach()** olarakgeçmektedir. ) ile bağlamalı ve bunlar üzerinde çeşitli manipülasyonlar yapmamız gerekmektedir.

metin, ekran, ekran görüntüsü, iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu

## Veritabanından Bir Kayıt Silme

Bir verinin silinebilmesi için Context.tabloismi.remove(silinecek\_item) denilerek ve ardından bu değişikliği veritabanına kaydetmek için context.SaveChanges() methodunu çağırarak gerçekleştirebiliriz.

void deletebyid (int id )

{

using (var context = new SQLİTECONTEXT())

{

var data = context.Products.Where(x => x.ID == id).FirstOrDefault();

if (data!=null)

{

context.Products.Remove(data);

context.SaveChanges();

}

}

}

Aynı şekilde el yordamı ile oluşturulmuş olan bir nesneyi de sildirebilirim. Update işlemi için Tracking (Attach) gerektirirken bu işlem için sadece id bilgisi olan bir nesneyi yollamamız yeterlidir.

using (var context2 = new SQLİTECONTEXT())

{

var p = new Product() { ID = 2};

//context2.Entry(p).State = EntityState.Deleted;

context2.Products.Remove(p);

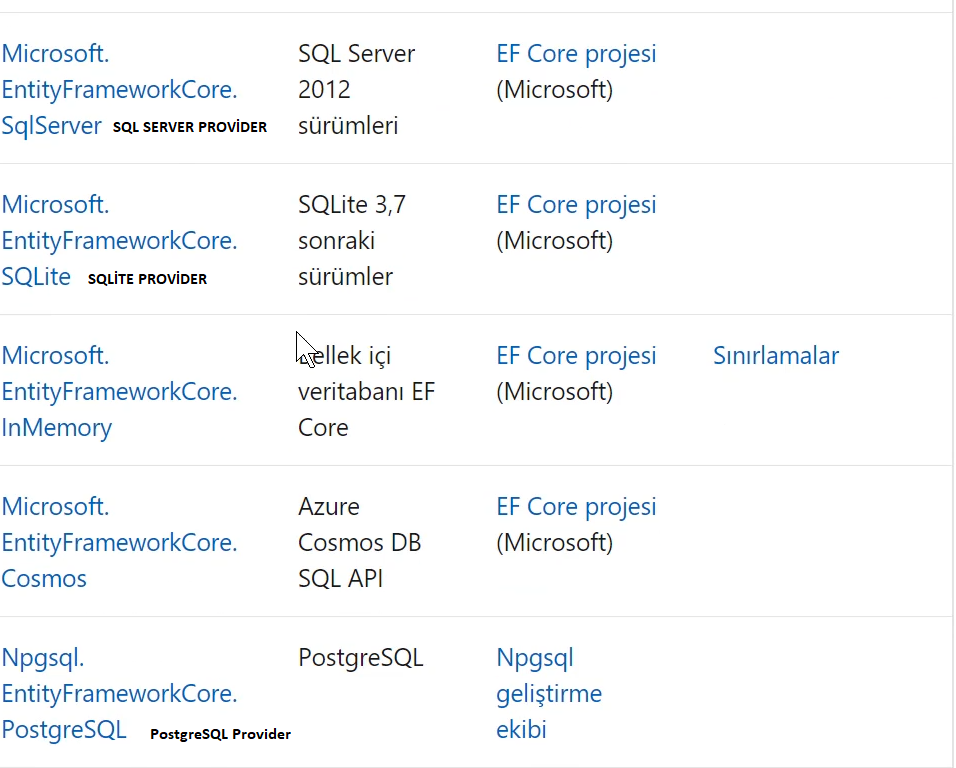
context2.SaveChanges();

}

Product remove yerine entry method aktif edilip remove method kaldırılarak ta işlem gerçekleştirilir. Esasındaki süreç Gelen verinin statüsünü belirlemektir. İki süreç te birbirlerinin aynısıdır.

## Provider Değişikliği İle Tüm Projeyi Çeşitli Veritabanları İçin Uygun Hale Getirmek

Önceki konu anlatımlarında da detaylarına çokça yer verilen Entity Framework yapısında hedef provider üzerinde yapılan bir değişiklik yapılan hedef sisteme göre entegre olabilmektedir. Bu süreçlerin gerçekleştirilmesinde dikkat edilmesi gereken önemli husus hedef provider in sistemde kurulu olmasıdır. Provider için Nuget platformu veya Microsoft.com üzerinden paketin tedarik edilmesi gerekmektedir.

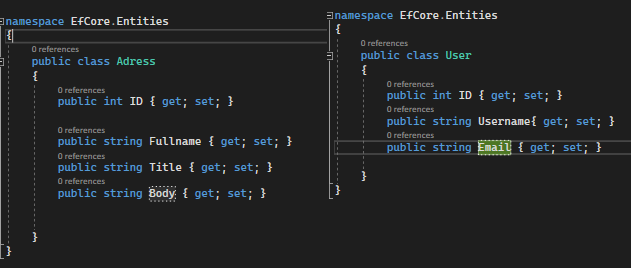


Paketler bir üstteki resimde gözükmektedir. Paket kurulması ile beraber Context sınıfından Override edilmiş olan onconfigüring methoduna giderek yen connection stringinin belirlenmesi yeterli olacaktır.

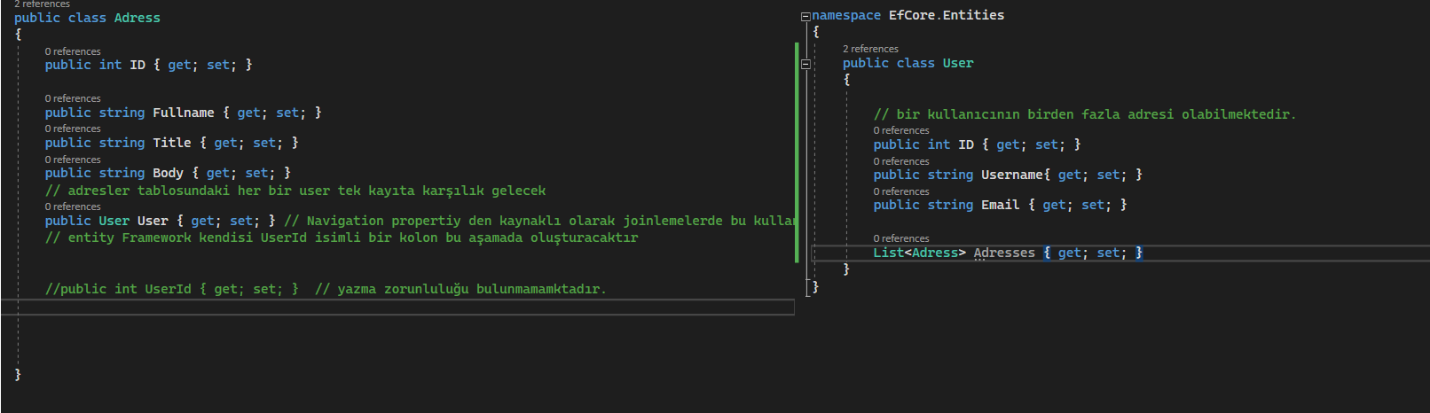
# ENTİTY FRAMEWORK CORE İLİŞKİLER

## One To Many (1-N)

Başlıktan da anlaşılacağı üzere 1 e N ilişkinin olduğu bir yapıdır. Bir siparişte birden çok ürün olabilir. Fakat her ürün bir siparişe ait olabilir ilişkisi düşünüldüğünde 1-N ilişkisi verilebilmektedir. Süreçi analiz edebilmek için adres ve kullanıcılar sınıflarını oluşturalım.



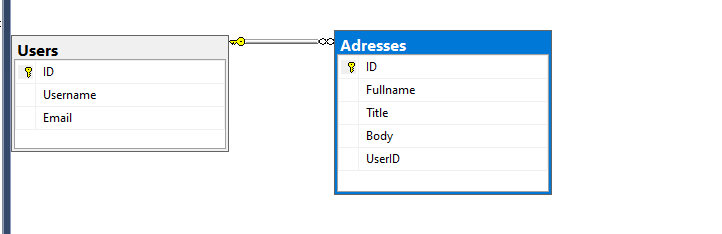
Bir kullanıcının birden fazla adresi olabileceği ama her adresin sadece bir kullanıcıya ait olduğu sistemi burada uygulayalım Userin Primary Key I Adress için Foreign Key Oluşturmalıdır. Bu açıdan bakıldığında :



Bu şekilde bir class oluşturmamız gerekmektedir. Bir kullanıcının birden fazla adresi olabilir bu sebeple generic bir liste olarak tanımlayıp içerisini dodlurtabiliriiz. Bir adresin ise birden fazla kullanıcısı bulunabilir. Bu koşul düşünüldüğünde sadece bir adet Kullanıcı nesnesi tanımlayabiliriz. Kayıtların eşleşebilmesi için ve foreign key oluşturabilmesi için

public int UserId { get; set; }

eklenmesi uygundur. Fakat objenin kendisi olduğundan ötürü bunu eklemesek dahi biz migration ımızı oluşturup bu migration üzerinden bir güncelleme işlemi yaptıgımız anda bizlere veritabanında Userid isimli bir değişken oluşturmaktadır. Bu işlem Navigation Property olarak adlandırılmaktadır. **Eklenmemiş** hali ile ilgili görsel bir aşağıda mevcuttur.



Foreign keyleri bağlarken nullable olabilmesi gibi yapılandırmalar için bu property I ekleyerek int? şekilde yapılandırabiliriz. Bu sayede sahibi olmayan adresleri de db ye yollamamız mümkündür.