STORE FLOW PROJESİ

**🔹 Microsoft.EntityFrameworkCore**

**EF Core’un ana kütüphanesidir.**  
Veritabanı işlemleri (CRUD – ekleme, silme, güncelleme, listeleme) için temel sınıflar ve altyapı buradadır.

Bu paket olmadan EF Core kullanılamaz.

**🔹 Microsoft.EntityFrameworkCore.Design**

**Tasarımsal işlemler** için kullanılır.  
Yani: migration (veritabanı şeması oluşturma/güncelleme) gibi komut satırı işlemleri için gereklidir.

EF Core’un CLI (komut aracı) özelliklerini destekler.

**🔹 Microsoft.EntityFrameworkCore.SqlServer**

**SQL Server için özel sürücüdür.**  
EF Core, farklı veritabanlarıyla çalışabilir. Biz SQL Server kullanacaksak bu paketi kurarız.

PostgreSQL veya MySQL için farklı sürümler vardır.

**🔹 Microsoft.EntityFrameworkCore.Tools**

Visual Studio’daki **PM Console (Package Manager Console)** üzerinden migration, update gibi işlemleri yapabilmek için gereklidir.

Komutları çalıştırmak için araçları içerir.

**💡 Özetle**

| **Paket Adı** | **Ne İşe Yarar?** |
| --- | --- |
| EntityFrameworkCore | EF Core’un temel yapısı |
| EntityFrameworkCore.Design | Migration gibi tasarım işlemleri |
| EntityFrameworkCore.SqlServer | SQL Server bağlantısı için |
| EntityFrameworkCore.Tools | Komut satırı işlemleri için |

ADIM ADIM GİDİLEN YOLLAR

* Entity Oluşumu için
* paketlerin kurulması

1. Microsoft.EntityFrameworkCore
2. Microsoft.EntityFrameworkCore.Design
3. Microsoft.EntityFrameworkCore.SqlServer
4. Microsoft.EntityFrameworkCore.Tools

paketleri kur sınıfları aç örnek vermek gerekirse



sonra dbcontext kur



Bire Çok İlişki Kurma



EF Core bunu okuyunca şunu anlar:

Bu ürün CategoryId üzerinden Category ile ilişkilidir.



Category bir **birden çoğa (one-to-many)** ilişkiyi temsil eder:

* Bir kategoriye birden fazla ürün (Product) ait olabilir.
* Products listesi, o kategoriye bağlı tüm ürünleri tutar.

ViewComponent Kurulumu

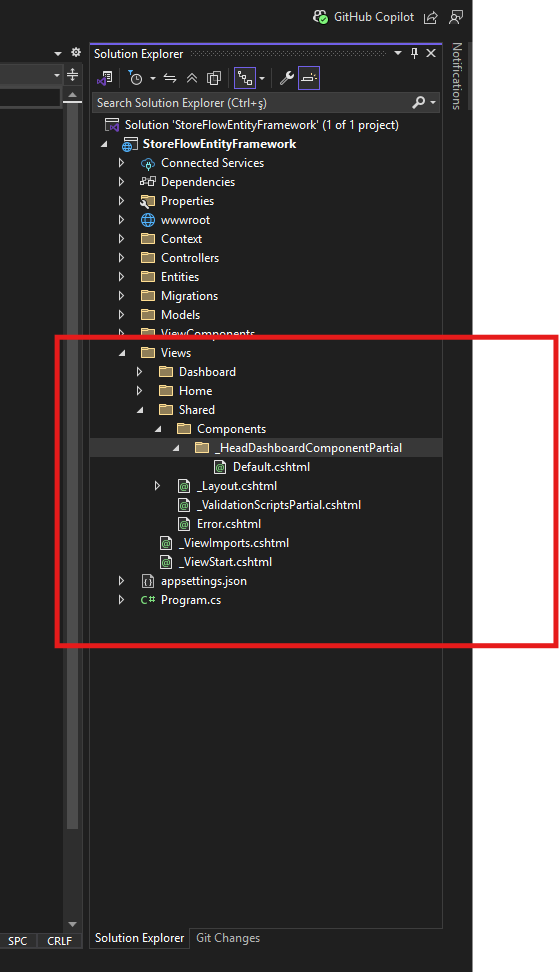
Klasör oluştur

bir class aç adı örnek vermek gerekirse

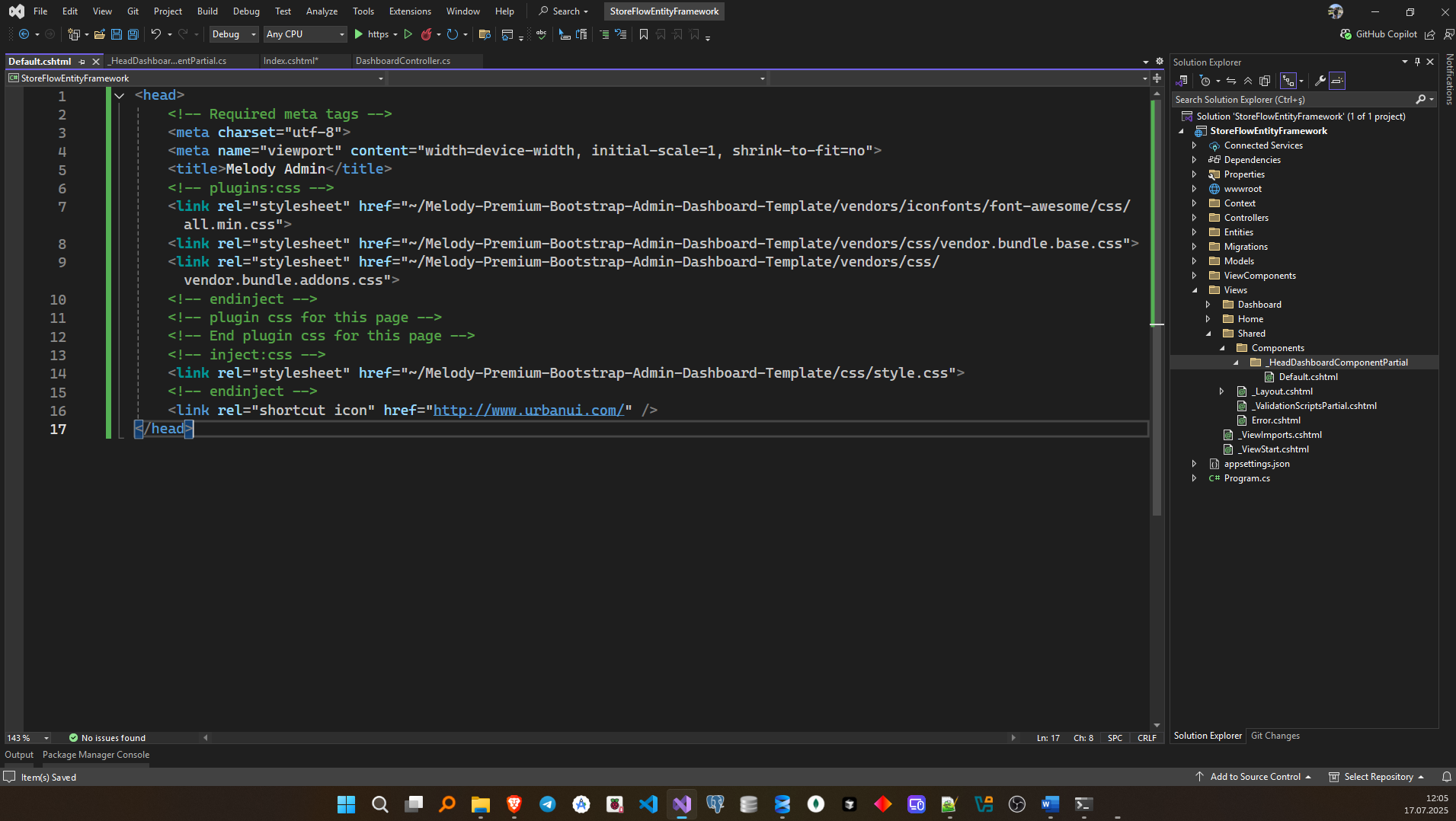
\_HeadDashboardComponentPartial gibi

classın içi böyle olacak

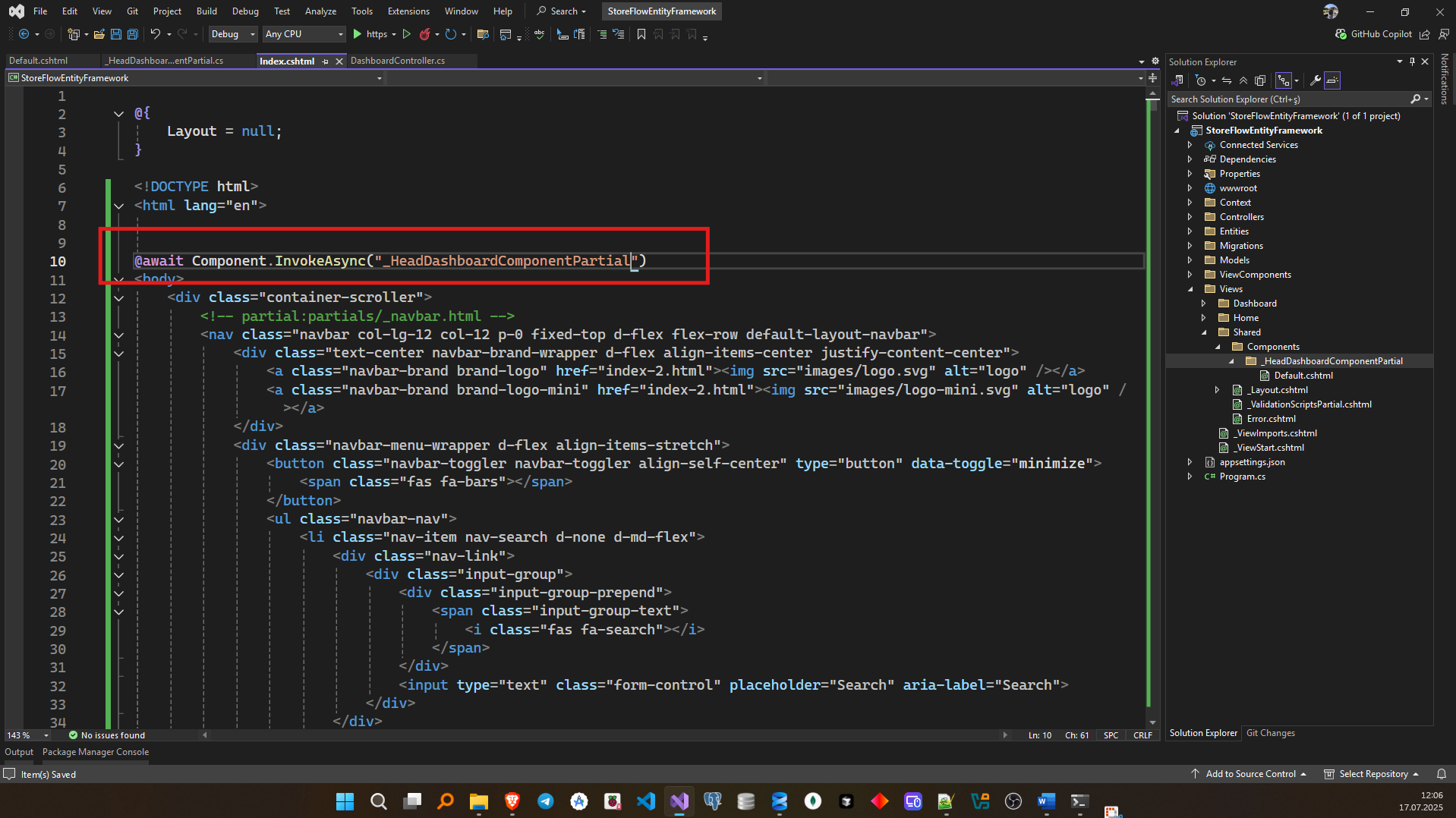




Sonra View Altında Shared Altında Components Altında Klasör Oluştur ve onada Clasımızın ismini ver sonra classdan da bir view oluştur Ve o default olmalı



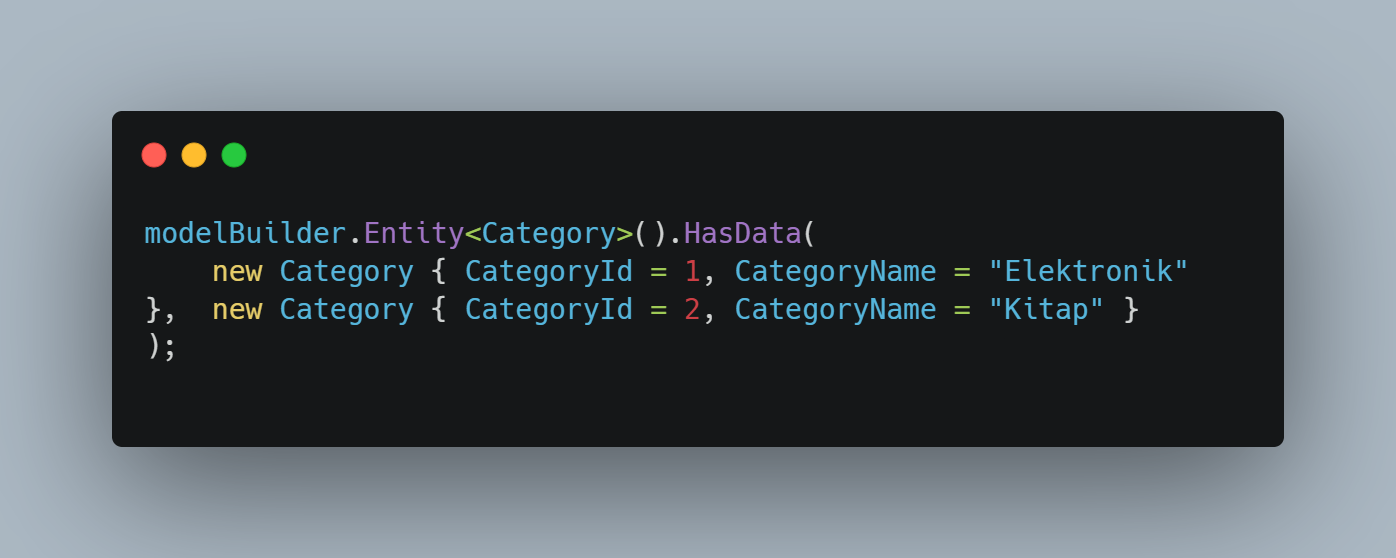
Sonra Alacağın kısmı kes ve Default adlı yere yapıştır



Kesilen Yere Yapıştır böylelikle O Kısmı Çağırabilelim

**🔹 Seed (Tohumlama) Nedir?**

* **Seed**, veritabanı ilk oluşturulduğunda içine **hazır veri eklemek** demektir.
* Genellikle DbContext içinde, OnModelCreating() veya migration sonrası HasData() ile kullanılır.



* Seed veriler migration sırasında veritabanına eklenir.

EF KOMUTLARI



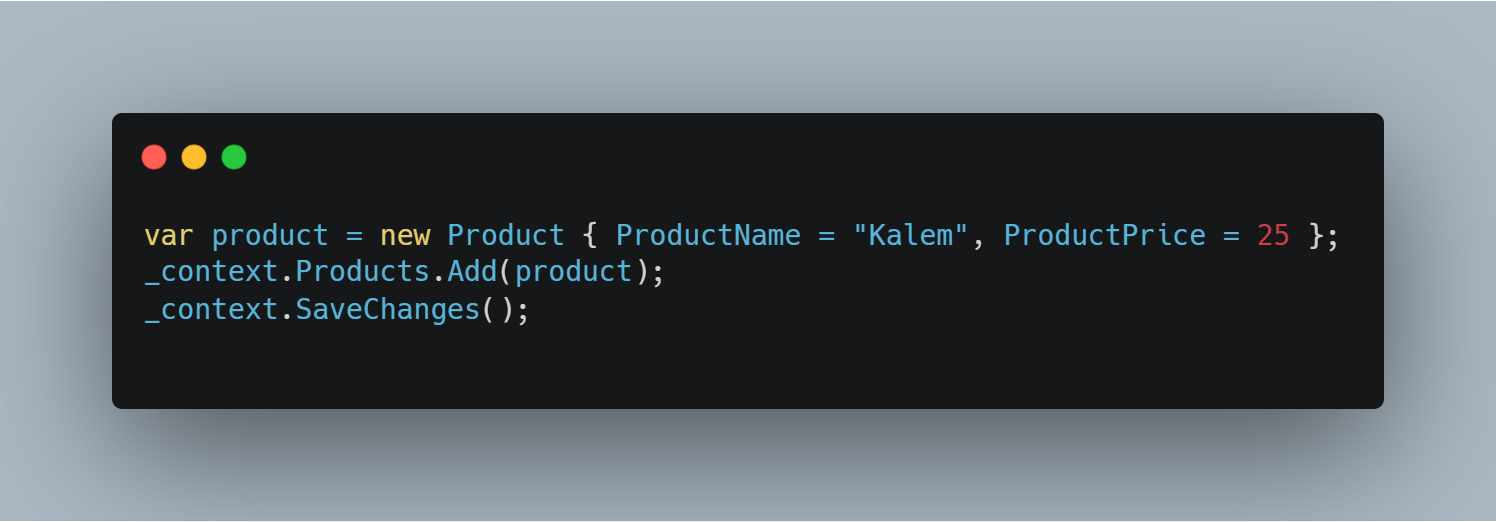
**✅ EF Core - AddAsync vs Update vs SaveChangesAsync Kısa Notları**

* AddAsync()  
  → Yeni veri belleğe alınır, asenkron şekilde eklemeye hazırlanır.  
  → **Veritabanına gitmez**, sadece hazırlar.  
  → Gerçek kayıt SaveChangesAsync() ile yapılır.
* Update()  
  → Mevcut veriyi "güncellenecek" olarak işaretler.  
  → **Veritabanına sorgu atmaz**, sadece izleme (tracking) yapılır.  
  → Asenkron gerekmez, o yüzden UpdateAsync() yoktur.
* SaveChangesAsync()  
  → Tüm ekleme, silme, güncelleme işlemlerini **veritabanına yazar.**  
  → Gerçek işlemi yapan yerdir.  
  → Asenkron çalışır, UI'ı kilitlemez.

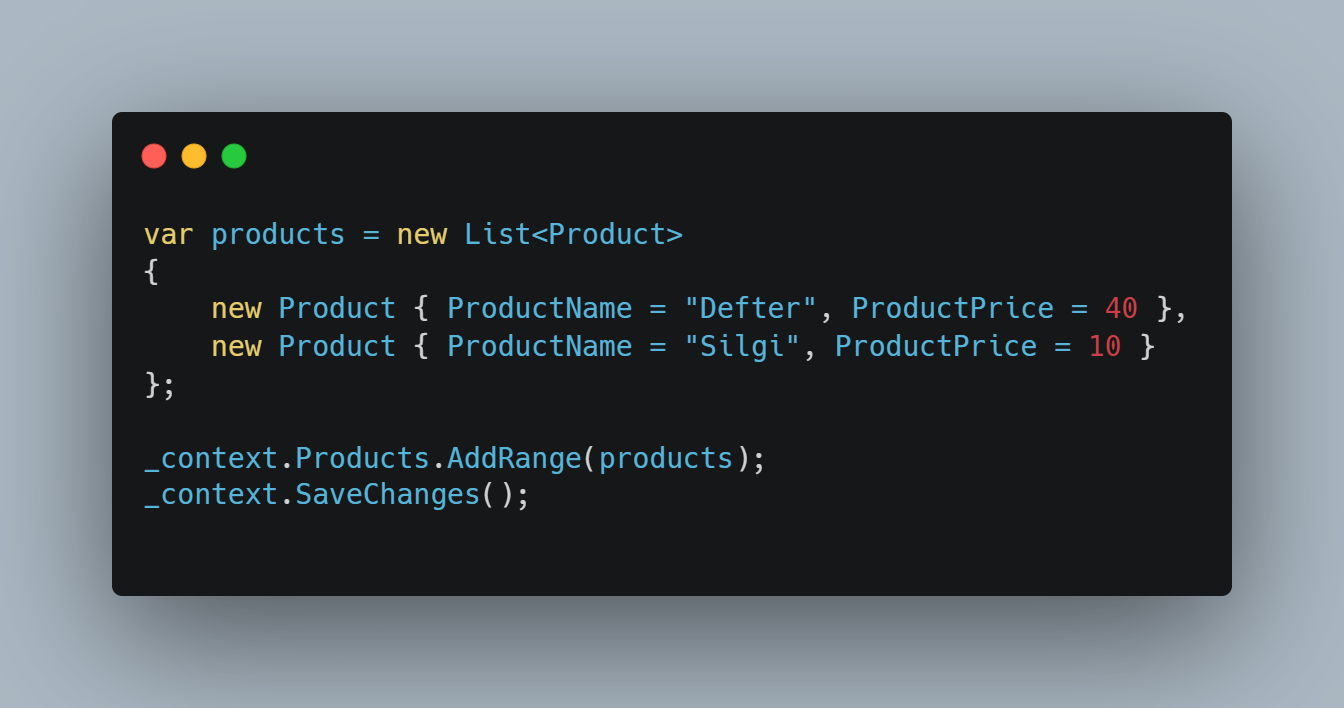
**🧠 Ekstra Bilgi:**

* Update() çoğu zaman **gerekmez**, EF Core zaten Find ile çekilen nesnelerde yapılan değişiklikleri otomatik takip eder.

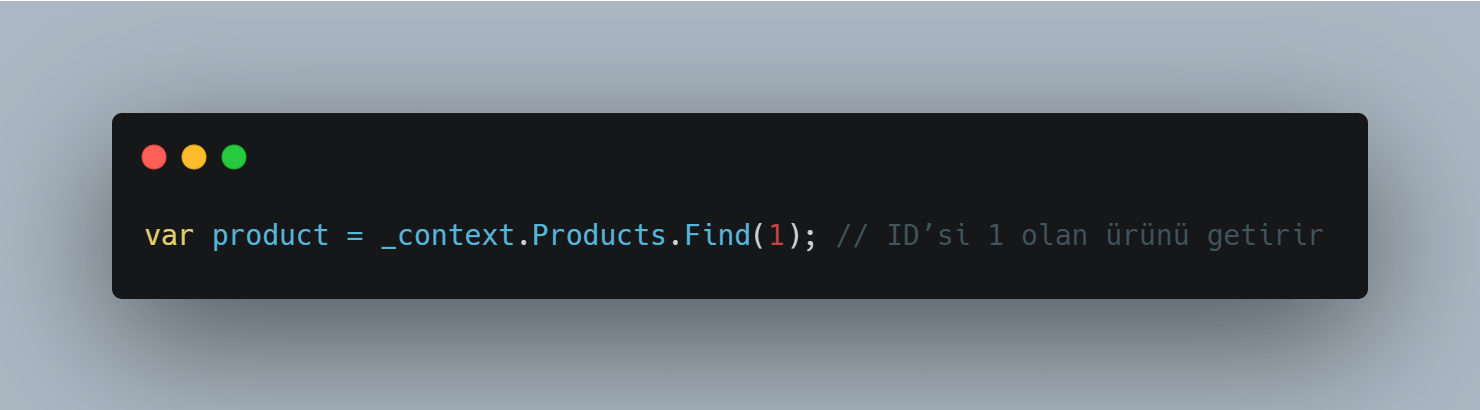
**🔹 1. Add() – Tek veri ekleme**



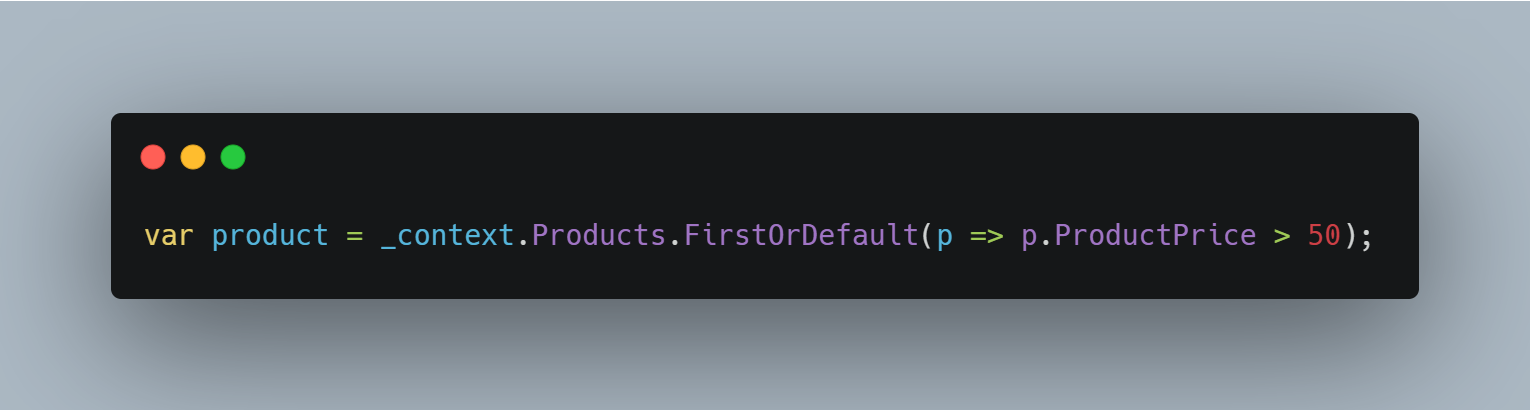
**🔹 2. AddRange() – Çoklu veri ekleme**



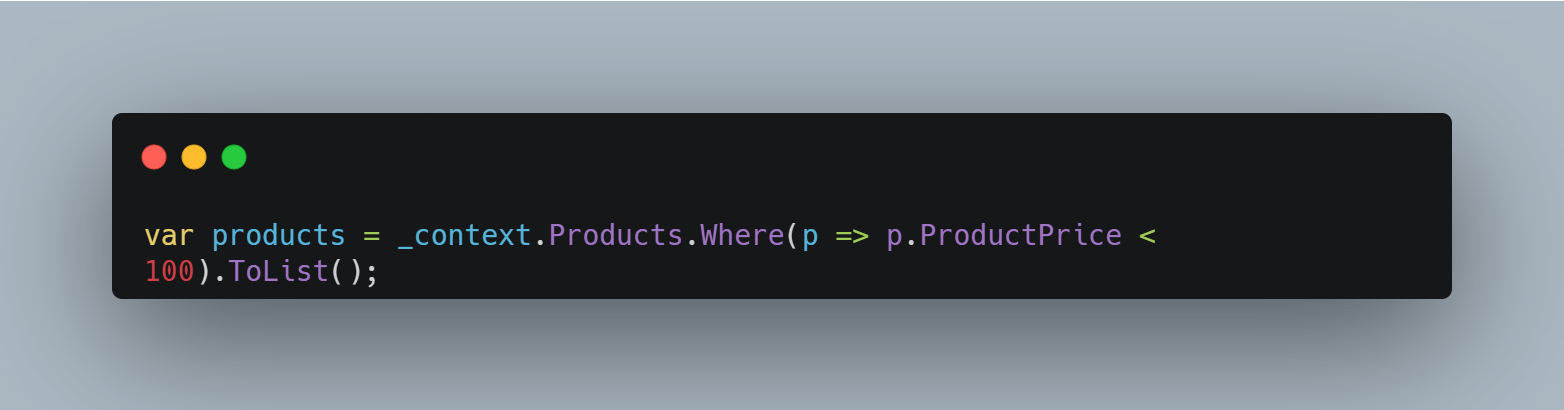
**🔹 3. Find() – ID ile veri getirme**



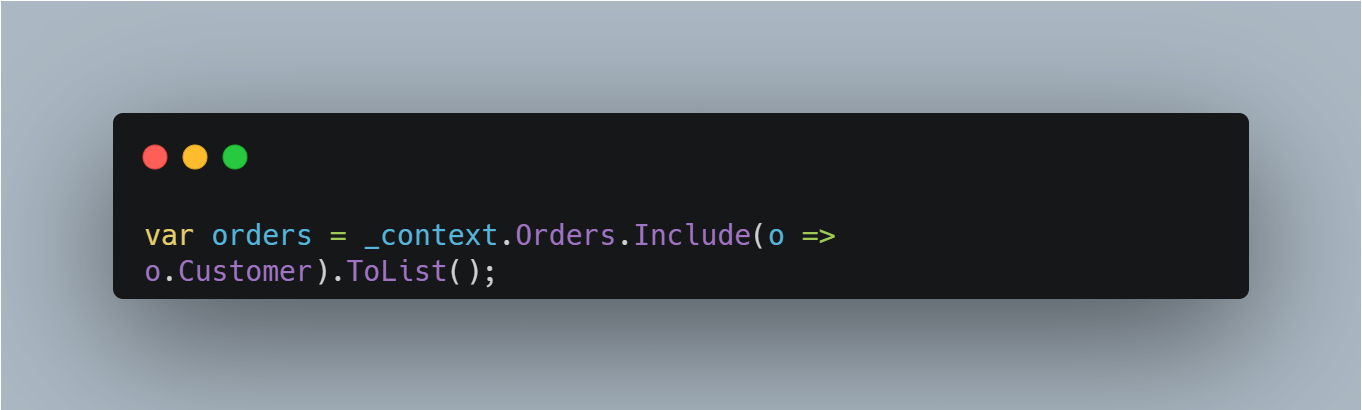
**🔹 4. FirstOrDefault() – İlk eşleşen veriyi getir**



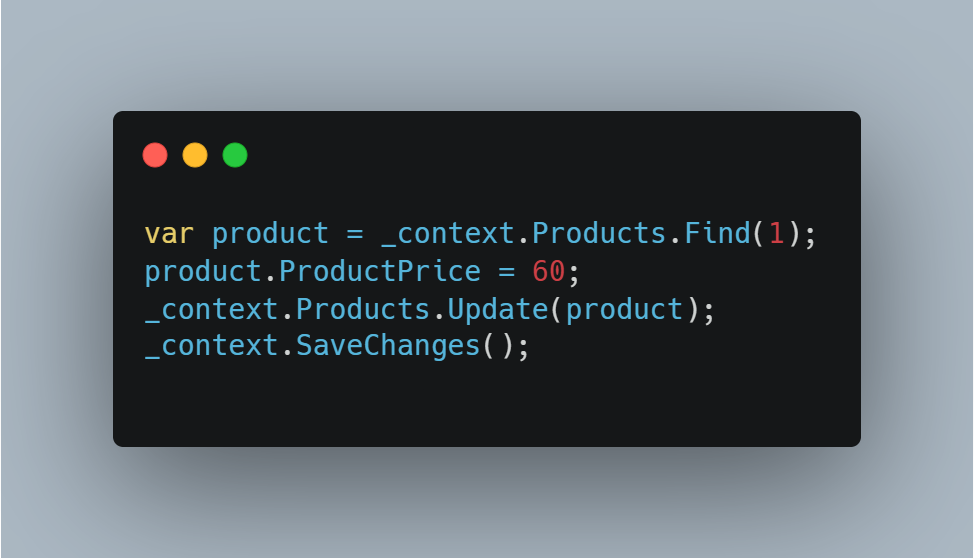
**🔹 5. Where() – Filtreleme**



**🔹 6. Include() – İlişkili tabloyu çekme**



**🔹 7. Update() – Veri güncelleme**



**🔹 8. Remove() – Tek veri silme**



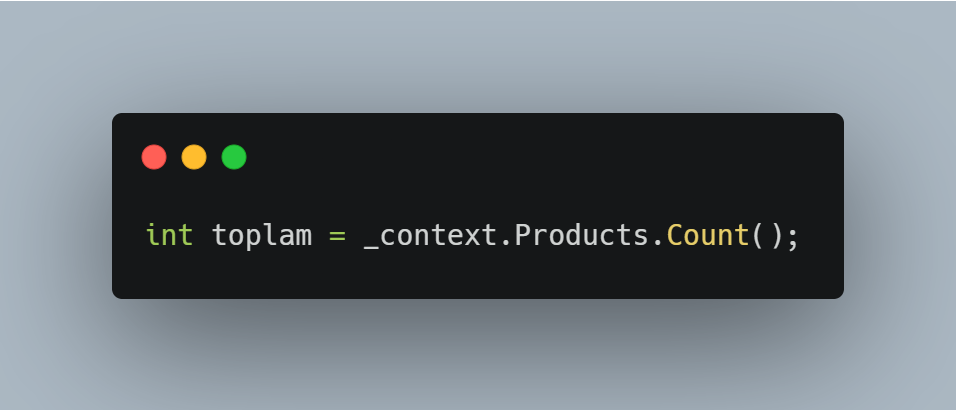
**🔹 9. RemoveRange() – Çoklu veri silme**



**🔹 10. Any() – Veri var mı?**



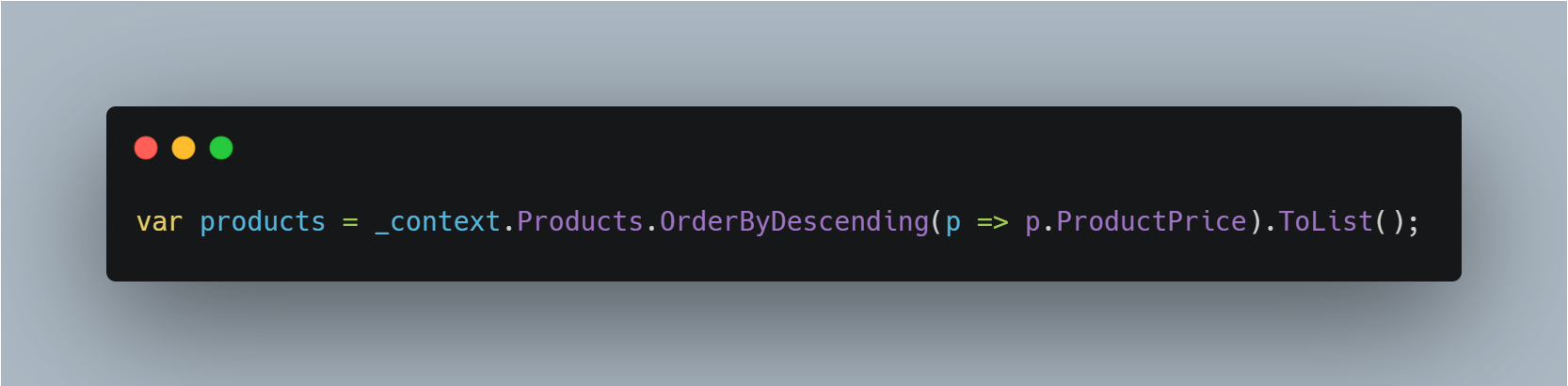
**🔹 11. Count() – Veri sayısı**



**🔹 12. OrderBy() – Artan sıralama**



**🔹 13. OrderByDescending() – Azalan sıralama**



**🔹 14. Min() ve Max() – En küçük / büyük değer**



**🔹 15. SaveChangesAsync() – Asenkron kayıt**



**🔹 16. AddAsync() – Asenkron veri ekleme**



**🔹 17. AddRangeAsync() – Asenkron çoklu ekleme**



| **Komut** | **Açıklama** |
| --- | --- |
| Add() | Tek veri ekler (belleğe alır) |
| AddAsync() | Tek veriyi asenkron ekler |
| AddRange() | Çoklu veri ekler |
| AddRangeAsync() | Çoklu veriyi asenkron ekler |
| Find(id) | Id’ye göre veri getirir |
| FirstOrDefault() | İlk eşleşen veriyi getirir |
| ToList() | Liste halinde getirir |
| Where() | Filtreleme yapar |
| Include() | İlişkili verileri getirir |
| Update() | Veriyi güncelleme için işaretler |
| Remove() | Tek veri siler |
| RemoveRange() | Çoklu veri siler |
| SaveChanges() | Bellekteki tüm değişiklikleri DB’ye yazar |
| SaveChangesAsync() | Asenkron olarak DB’ye yazar |
| Any() | Veri var mı diye kontrol eder |
| Count() | Kaç veri olduğunu sayar |
| Min() / Max() | En küçük / en büyük değeri getirir |
| OrderBy() | Artan sıralama yapar |
| OrderByDescending() | Azalan sıralama yapar |

**🔍 Filtreleme (Filtering)**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| Where | Şarta uyan kayıtları getirir. | db.Products.Where(p => p.Price > 100) |
| Any | En az bir kayıt şartı sağlıyorsa true döner. | db.Products.Any(p => p.Stock == 0) |
| All | Tüm kayıtlar şartı sağlıyorsa true döner. | db.Products.All(p => p.Stock > 0) |
| Contains | Koleksiyon ya da string içinde arama yapar. | db.Products.Where(p => p.Name.Contains("kalem")) |

**🔢 Sıralama (Ordering)**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| OrderBy | Artan şekilde sıralar. | db.Products.OrderBy(p => p.Price) |
| OrderByDescending | Azalan şekilde sıralar. | db.Products.OrderByDescending(p => p.Stock) |
| ThenBy | İkincil sıralama kriteri ekler. | db.Products.OrderBy(p => p.Price).ThenBy(p => p.Name) |

**🎯 Projeksiyon (Seçme / Dönüştürme)**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| Select | Belirli alanları çeker veya yeni model oluşturur. | db.Products.Select(p => p.Name) |
| SelectMany | İlişkili koleksiyonları tek düz listeye indirger. | db.Categories.SelectMany(c => c.Products) |

**➕ Ekleme / Güncelleme / Silme**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| Add | Tek bir kayıt ekler. | db.Products.Add(new Product { ... }) |
| AddRange | Çoklu kayıt ekler. | db.Products.AddRange(listOfProducts) |
| Update | Kayıt günceller. | db.Products.Update(product) |
| Remove | Kayıt siler. | db.Products.Remove(product) |
| RemoveRange | Çoklu kayıt siler. | db.Products.RemoveRange(list) |

**🧮 Sayma ve Toplama**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| Count() | Kayıt sayısını döner. | db.Products.Count() |
| LongCount() | Çok büyük veri setleri için kullanılır. | db.Products.LongCount() |
| Sum | Alanın toplamını alır. | db.Products.Sum(p => p.Price) |
| Average | Ortalama değeri döner. | db.Products.Average(p => p.Price) |
| Min, Max | Minimum / maksimum değer. | db.Products.Min(p => p.Price) |

**⛓️ Gruplama (Grouping)**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| GroupBy | Veriyi belirli bir kritere göre gruplar. | db.Products.GroupBy(p => p.CategoryId) |

**📄 İlk / Son / Tekil Kayıt**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| First, FirstOrDefault | İlk kaydı döner. | db.Products.First(p => p.Stock > 0) |
| Last, LastOrDefault | Son kaydı döner (InMemory veya sıralı listelerde). | db.Products.LastOrDefault() |
| Single, SingleOrDefault | Tek bir kayıt bekler. | db.Products.Single(p => p.Id == 5) |

**🔁 Sayfalama (Paging)**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| Skip(n) | İlk n kaydı atlar. | db.Products.Skip(10) |
| Take(n) | n kadar kayıt getirir. | db.Products.Take(5) |
| Skip(...).Take(...) | Sayfalama için ideal kullanım. | db.Products.Skip(10).Take(10) |

**🔗 Join ve Include (İlişkilendirme)**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| Include | İlişkili tabloyu dahil eder (Eager Loading). | db.Products.Include(p => p.Category) |
| ThenInclude | İç içe ilişkileri dahil eder. | db.Orders.Include(o => o.Customer).ThenInclude(c => c.Address) |
| Join | Manuel join işlemi yapar. | db.Products.Join(db.Categories, p => p.CategoryId, c => c.Id, (p,c) => new { p.Name, c.Name }) |

**🧪 Koşullar ve Null Kontrolleri**

| **Metot** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| DefaultIfEmpty() | Boş koleksiyonlarda null yerine varsayılan değer döner. | db.Products.DefaultIfEmpty() |
| Distinct() | Tekrarlayan kayıtları ayıklar. | db.Products.Select(p => p.Name).Distinct() |

**🔁 Aggregate**

**Tanım**: Bir koleksiyondaki elemanları belirli bir işlemle tek bir sonuca indirger (akümülasyon yapar).

**Kullanım**: Toplama, çarpma, string birleştirme gibi işlemler için kullanılır.



**➕ Append**

**Tanım**: Koleksiyonun sonuna yeni bir eleman ekler (orijinal koleksiyonu değiştirmez, yeni koleksiyon döner).



**⬅️ Prepend**

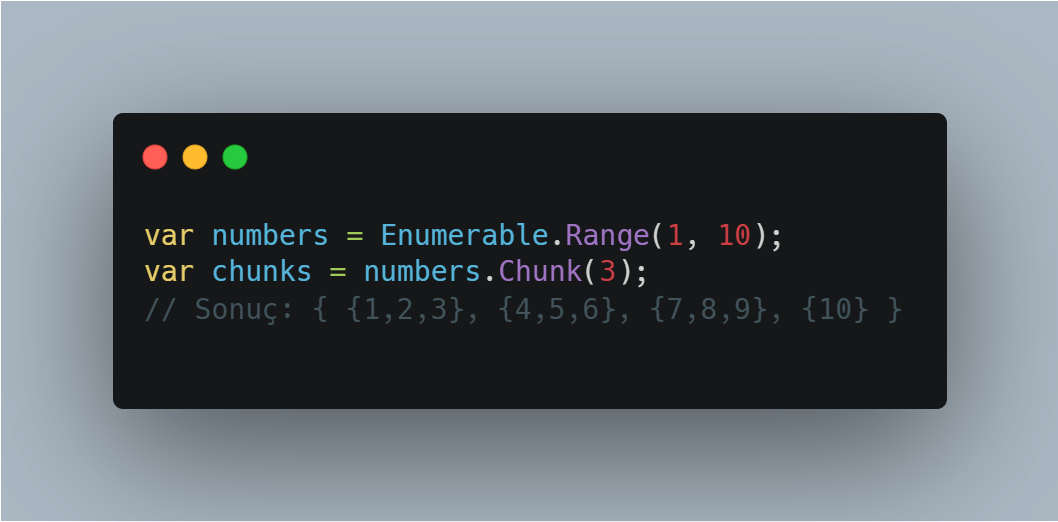
**Tanım**: Koleksiyonun başına yeni bir eleman ekler.



**🧩 Chunk**

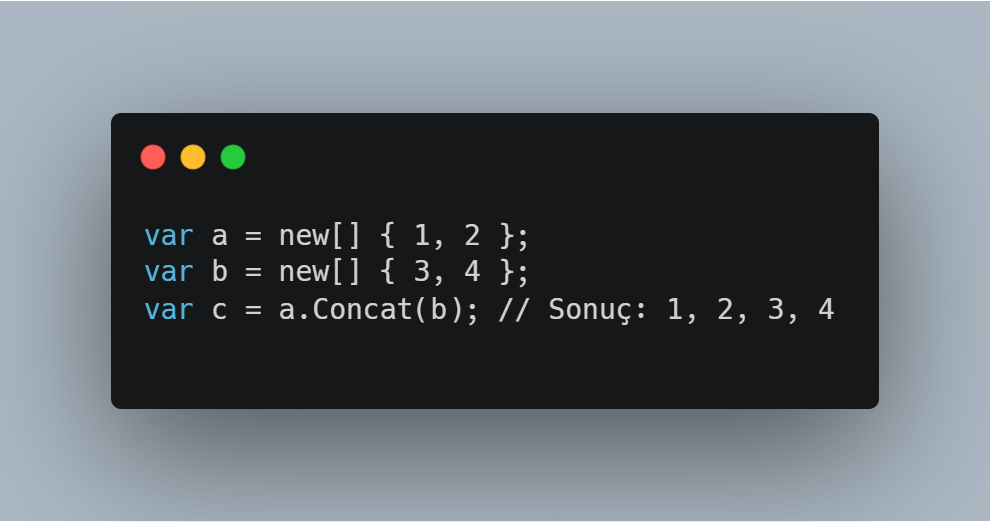
**Tanım**: Koleksiyonu eşit boyutlu parçalara böler (chunk = parça).

Chunk metodu .NET 6 ve sonrasında kullanılabilir.



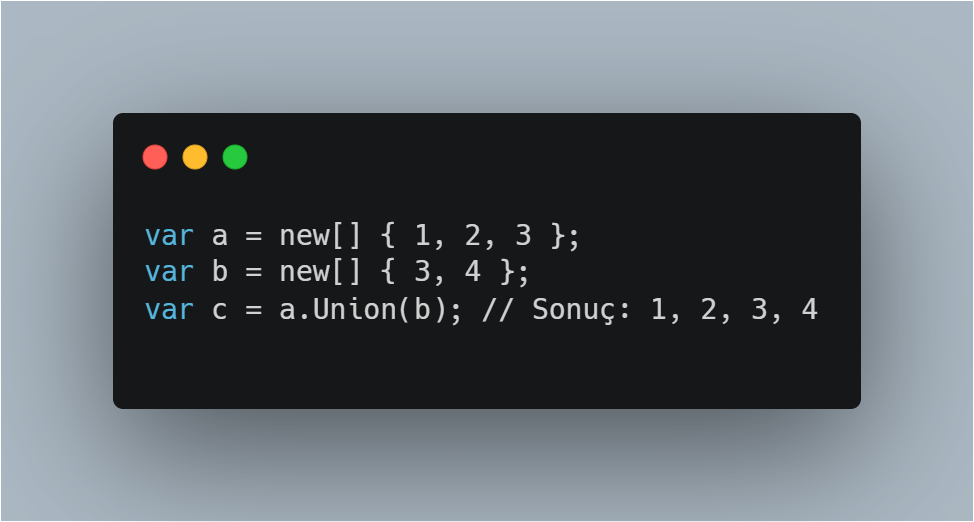
**🔗 Concat**

**Tanım**: İki koleksiyonu birleştirir (aynı türde olmalı), tekrarları **ayırt etmez**.



**🔄 Union**

**Tanım**: İki koleksiyonu birleştirir ama **tekrar edenleri atar** (Distinct gibi çalışır).

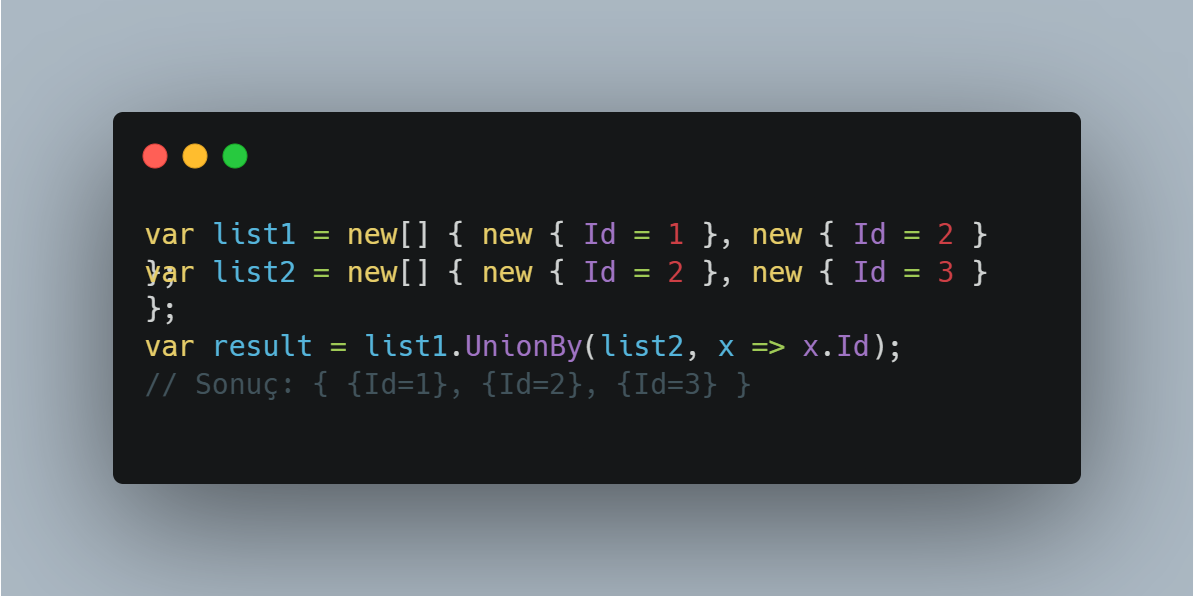


Concat → tekrarları alır  
Union → tekrarları ayıklar (set gibi davranır)

**🧠 UnionBy**

**Tanım**: Belirli bir özelliğe göre Union işlemi yapar.

.NET 6 ve üzeri.

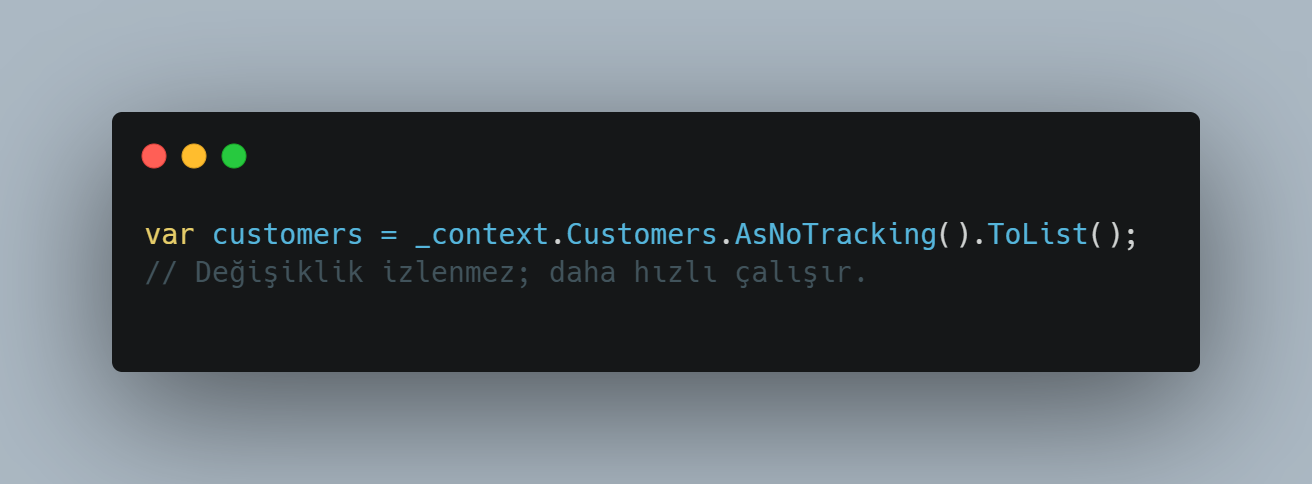


**📝 Özet Tablo**

| **Metot** | **Açıklama** | **Tekrarları Ayıklar** | **.NET Gereksinimi** |
| --- | --- | --- | --- |
| Aggregate | Koleksiyonu tek bir sonuca indirger | – | Tüm sürümler |
| Append | Koleksiyonun sonuna eleman ekler | – | .NET Core 3.0+ |
| Prepend | Koleksiyonun başına eleman ekler | – | .NET Core 3.0+ |
| Chunk | Koleksiyonu parçalara böler | – | .NET 6+ |
| Concat | Koleksiyonları birleştirir (tekrarlarla) | ❌ | Tüm sürümler |
| Union | Tekrarsız birleşim yapar | ✅ | Tüm sürümler |
| UnionBy | Özelliğe göre tekrarsız birleşim | ✅ | .NET 6+ |
|  |  |  |  |

**📌 AsNoTracking**

**Tanım:** EF Core'da veriyi izlemeye almaz; performansı artırır, güncelleme yapılmayacak sorgularda kullanılır.



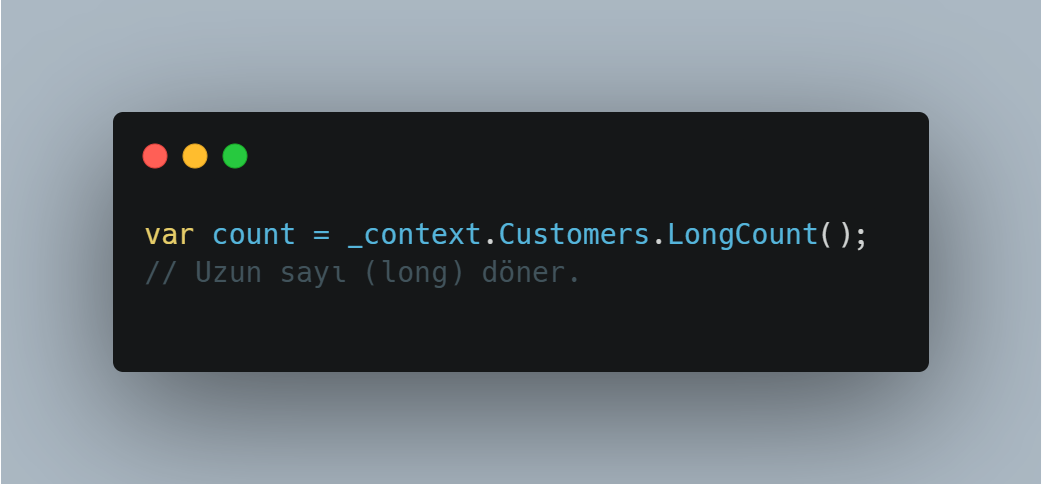
**📌 Attach**

**Tanım:** Varlığı (entity) izlenmek üzere EF Core context’ine dahil eder ama durumu "Unchanged" olur.



**📌 LongCount**

**Tanım:** Eleman sayısını long türünde döner. Çok büyük koleksiyonlar için kullanılır.



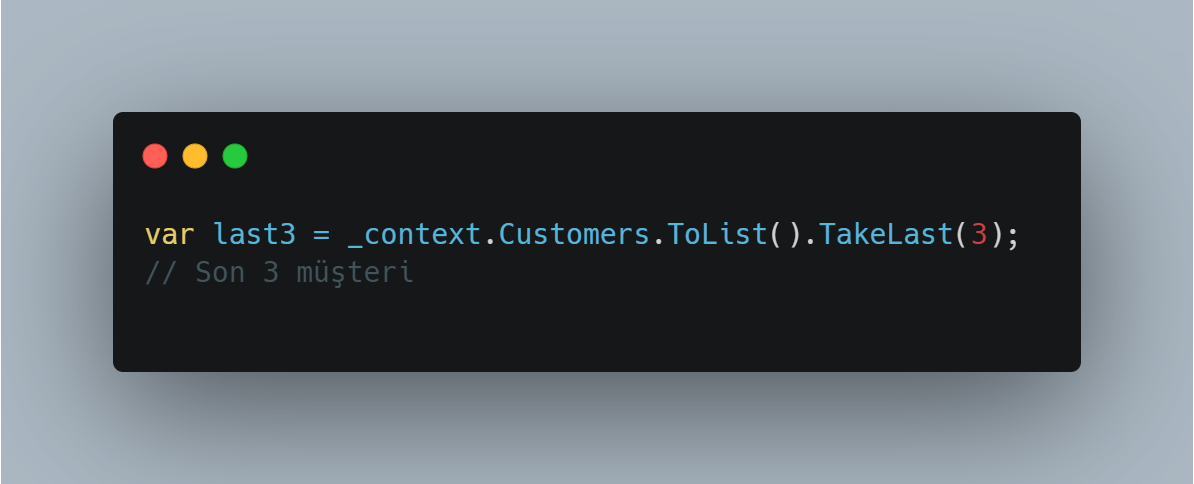
**📌 Last**

**Tanım:** Şarta uyan son elemanı verir. Sıralama yapılmazsa hataya neden olabilir.



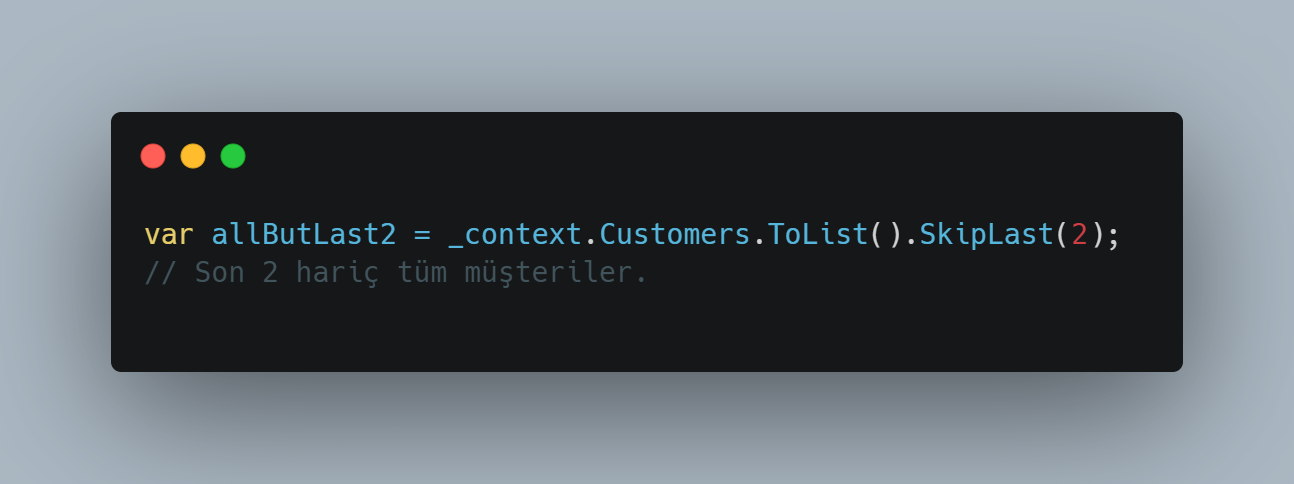
**📌 TakeLast**

**Tanım:** Listenin sonundaki belirli sayıda elemanı alır (koleksiyon hafızada olmalı).

.

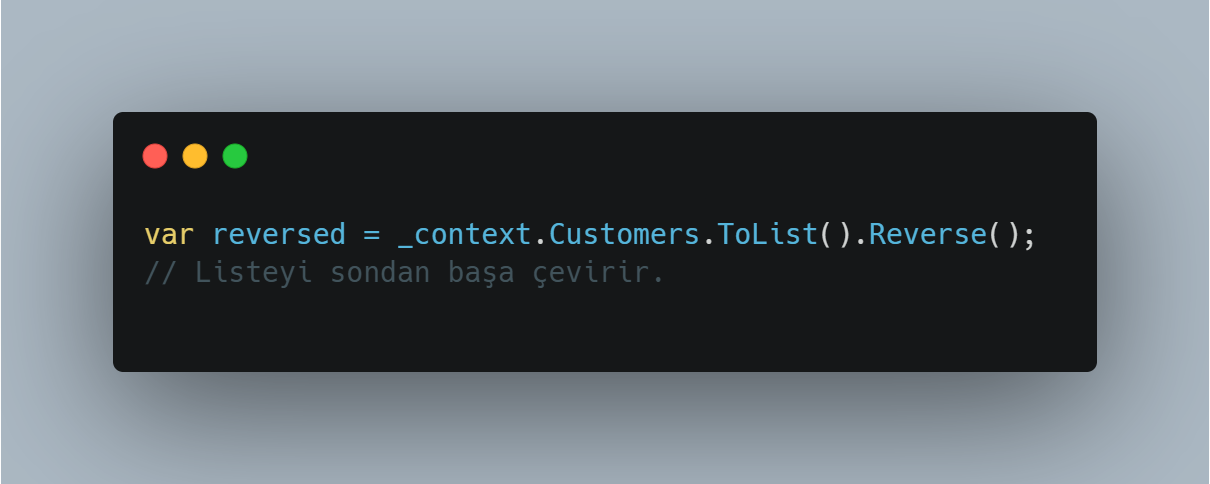
**📌 SkipLast**

**Tanım:** Listenin sonundaki belirtilen sayıda elemanı atlar (koleksiyon hafızada olmalı).



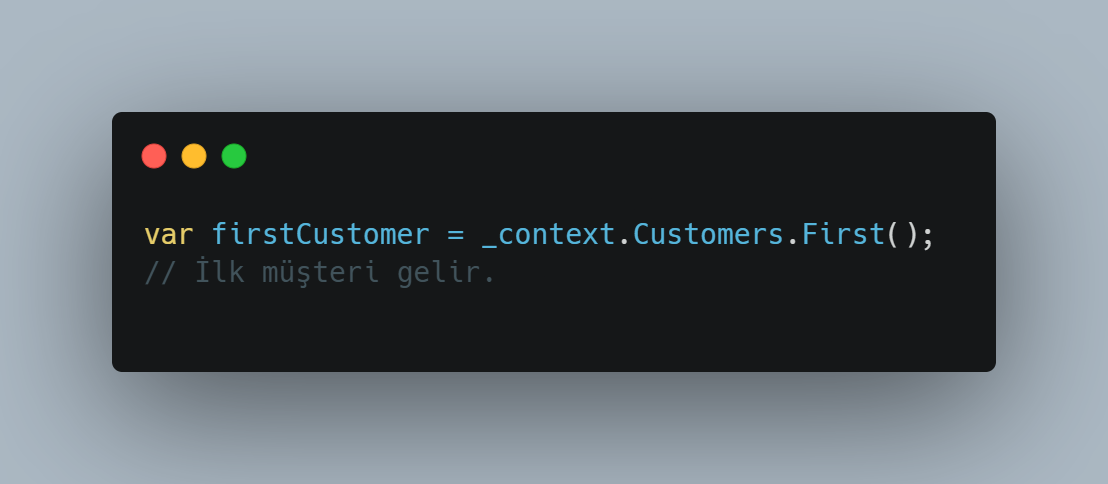
**📌 Reverse**

**Tanım:** Koleksiyonu ters çevirir.



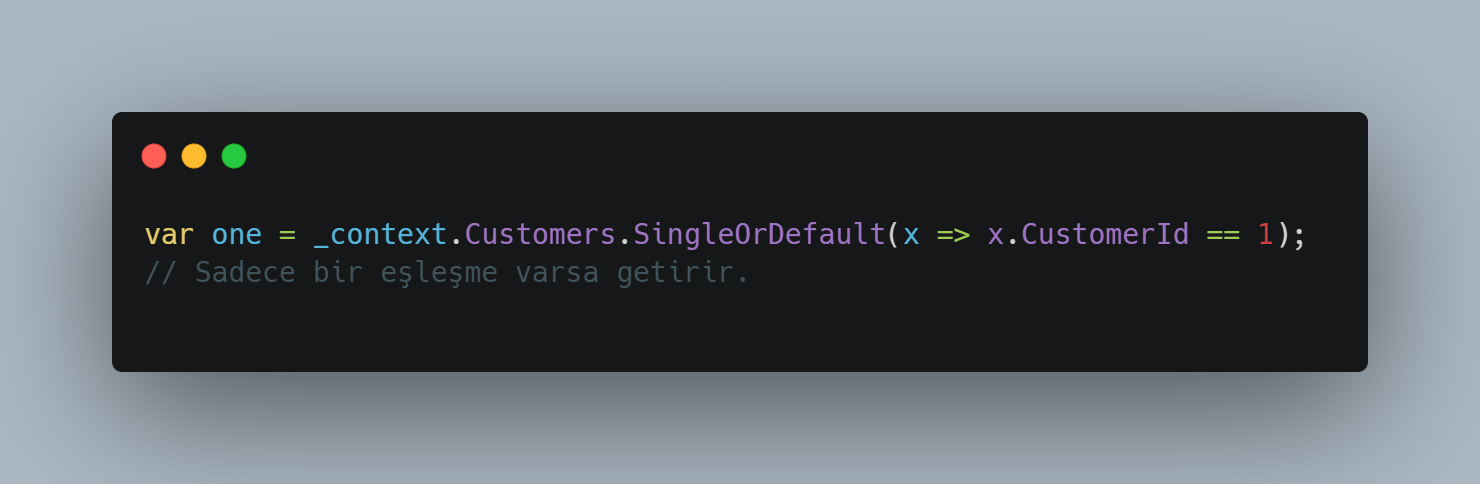
**📌 First**

**Tanım:** İlk elemanı döner; yoksa hata verir.



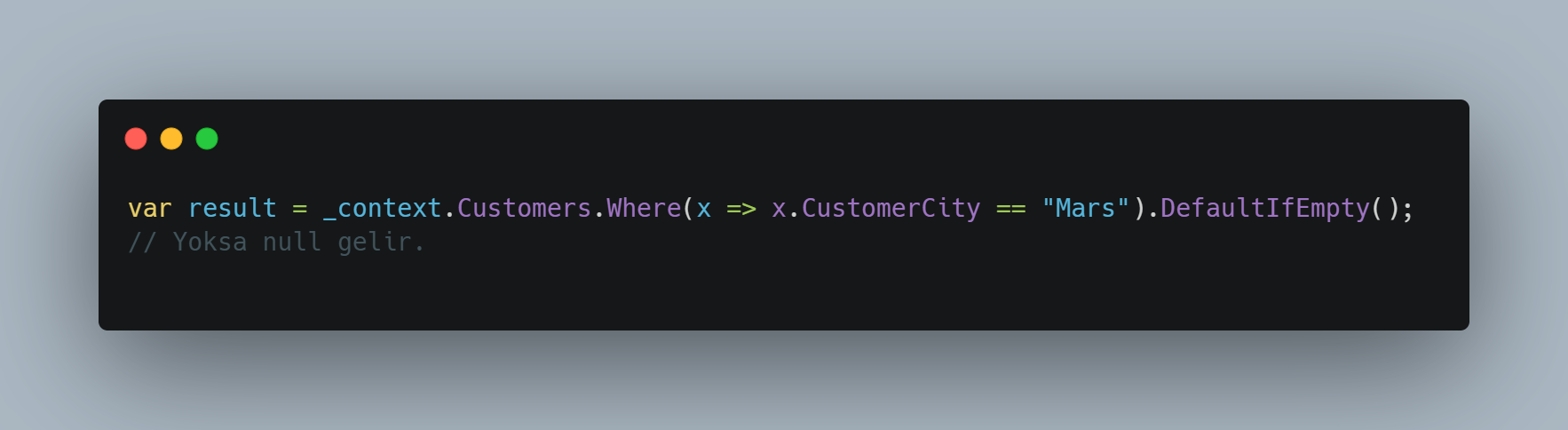
**📌 SingleOrDefault**

**Tanım:** Sadece bir eleman varsa getirir; yoksa null, birden fazlaysa hata.



**📌 DefaultIfEmpty**

**Tanım:** Liste boşsa varsayılan değeri döner.



**📌 Intersect**

**Tanım:** İki koleksiyondaki ortak elemanları verir.



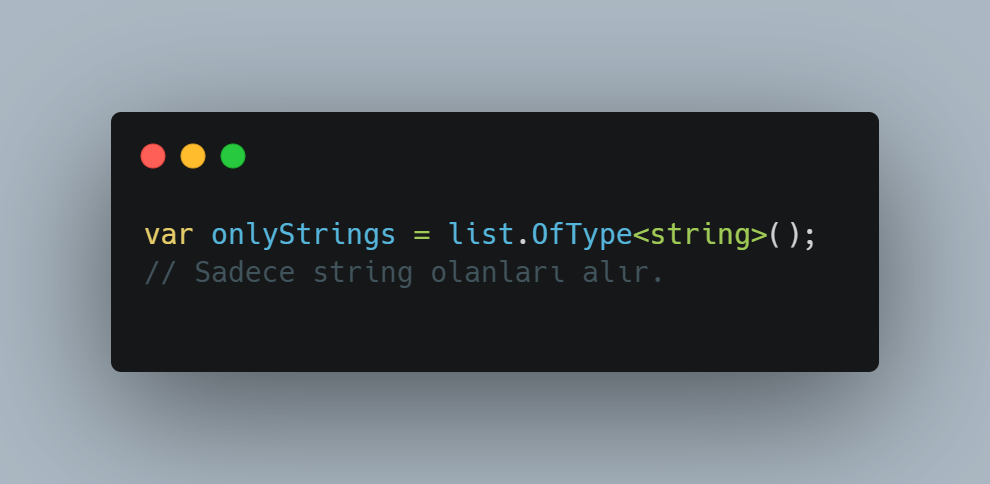
**📌 Cast<T>()**

**Tanım:** Elemanları belirli bir türe dönüştürür.



**📌 OfType<T>()**

**Tanım:** Belirli türdeki elemanları filtreler.



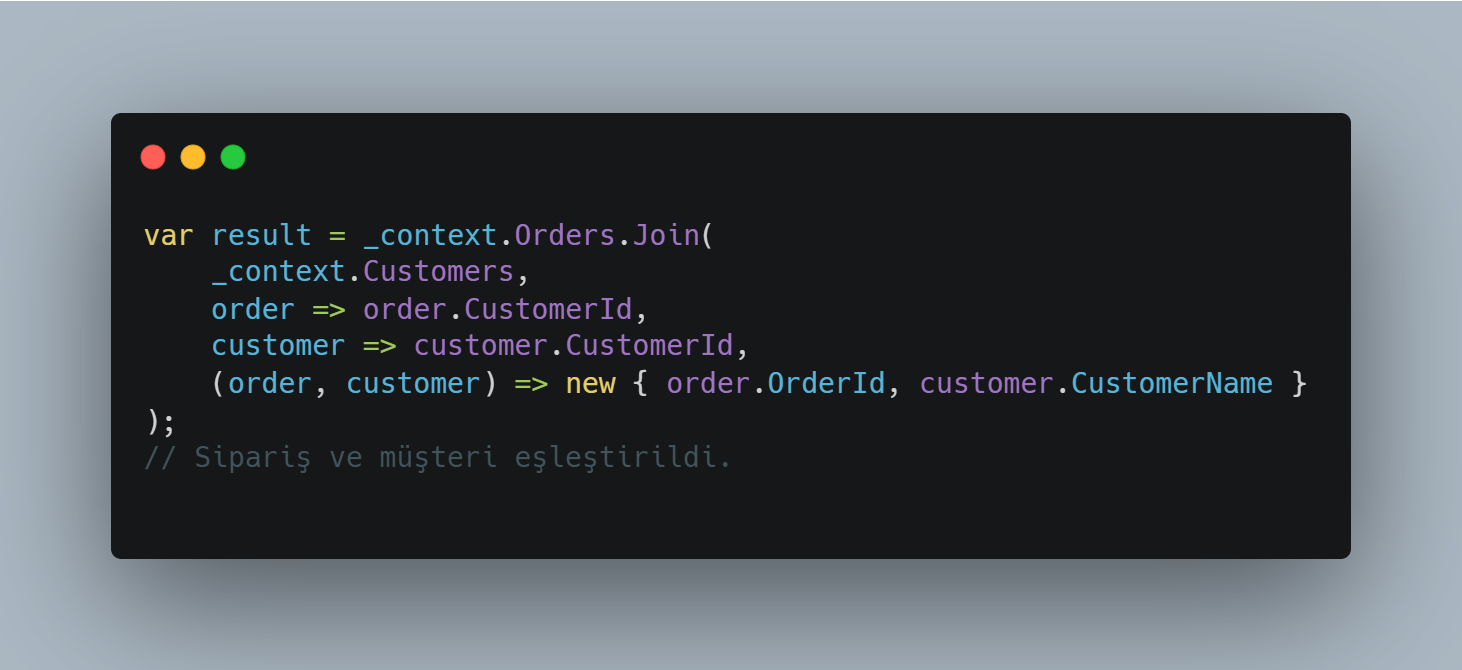
**📌 Entry**

**Tanım:** EF Core'da bir nesnenin (entity) durumuna erişmek için kullanılır.



**📌 Join**

**Tanım:** İki koleksiyonu birleştirir (SQL’deki INNER JOIN gibi).



**📌 Range**

**Tanım:** Belirli bir sayı aralığında liste oluşturur.



ÖNEMLİDİR

 **Include**: Zaten var olan entity ilişkilerini takip ederek **ilişkili nesneleri “otomatik” yükler**.

 **Join**: İki tabloyu veya koleksiyonu birleştirir ve **yeni şekillendirilmiş veri seti üretir**.

**🔹 IGrouping<TKey, TElement> Nedir?**

IGrouping, LINQ’de group by kullandığında dönen veri tipidir.

* TKey: Hangi özelliğe göre gruplandığını gösterir.
* TElement: O grubun içinde yer alan öğeleri gösterir.

Yani: **Bir anahtar (key)** ve o key’e ait **birden çok öğe (collection)**.

**📦 Örnek Senaryo:**

Elimizde bir Product listesi var:



**🔸 LINQ ile group by yapalım:**

var grouped = products.GroupBy(p => p.Category);

* grouped değişkeninin tipi: IEnumerable<IGrouping<string, Product>>
* Yani:
  + TKey: Category (örnek: "Elektronik")
  + TElement: Product (grubun içindeki ürünler)

**🔍 foreach ile nasıl yazılır?**



**🖨️ Çıktı:**

Kategori: Elektronik

--> Laptop

--> Telefon

Kategori: Giyim

--> Tişört

--> Pantolon

**🔧 Kısaca Özet:**

| **Terim** | **Anlamı** |
| --- | --- |
| GroupBy | Belirli bir alana göre gruplar oluşturur |
| IGrouping | Her bir grubun temsilidir |
| Key | Grup başlığıdır (örneğin: kategori adı) |
| TElement | Grup içindeki elemanlardır (örneğin: ürünler) |

**🎯 Ne zaman kullanılır?**

* Kategorilere göre ürünleri listelemek
* Departmanlara göre çalışanları gruplamak
* Tarihe göre satışları gruplamak
* Aylık, yıllık raporlama

**StringComparison.OrdinalIgnoreCase Nedir?**

* StringComparison enum (sıralı sabitler kümesi) içinde yer alan bir değer.
* Ordinal demek: **Karakterlerin Unicode (sayısal) değerlerine göre karşılaştırılması**.
* IgnoreCase demek: **Büyük/küçük harf duyarsızlığı** (case-insensitive).

**Yani**

StringComparison.OrdinalIgnoreCase, **iki string’i karşılaştırırken**:

* Harflerin büyüklüğünü/küçüklüğünü **önemsemez**,
* Doğrudan karakterlerin Unicode değerine göre hızlıca karşılaştırma yapar.