**ORM Nedir?**

ORM (Object Relational Mapping) veritabanı tablolarını ve ilişkilerini nesnelerle eşleştirerek ve bu nesneler üzerinden iletişim kurmayı esas alan yaklaşımdır.

Yani veritabanını doğrudan yazılım projesi içerisinde işlemek yerine oluşturulan nesneler üzerinden işlemeyi amaçlar.

**EFCore Nedir?**

ORM yaklaşımını benimsemiş bir araçtır. Kod içerisinde OOP ile SQL sorguları oluşturmamızı sağlar. Tekrarlı SQL sorgucuklarından bizleri kurtarır. Code First ve Database First yaklaşımları eşliğinde veritabanı ile yazılım arasındaki koordinasyonu sağlar.

**EFCore Nasıl Yüklenir?**

CLI

* cmd 🡺 **dotnet tool install --global dotnet-ef**
* Tool yüklendikten sonra herhangi bir projede kullanabilmek için **Microsoft.EntityFrameworkCore.Design** paketinin projede yüklü olması gerekmektedir.

PMC

* Visual Studio, PMC üzerinden talimatlar verebilmek için **Microsoft.EntityFrameworkCore.Tools** paketinin ilgili projede yüklü olması gerekmektedir.

**DBFirst ve CodeFirst**

EFCore, veritabanı çalışmaları için, veritabanının önceden olup olmaması durumlarına göre farklı yaklaşımlar sunar. Dolayısıyla hedef veritabanının kod kısmına aktarılması gerekmektedir. Lakin hedef veritabanın önceden oluşturulmuş mu yoksa yeni mi oluşturulacağı önem arz etmektedir. Sonuç olarak EFCore’un bize sunmuş olduğu yaklaşımlar veritabanının var olup olmaması durumlarına göre kod kısmında modellenme süreçlerinin hangi davranışla gerçekleştirileceğini belirleyecek olan tekniği bizlere sunmaktadır.

Veritabanı var olan bir projedeki yaklaşım büyük ihtimalle DBFirst, veritabanı inşa edilmemiş bir projede DBFirst ya da CodeFirst yaklaşımları tercih edilir.

Yani bir projede hangi yaklaşımın benimseneceği birkaç unsura bağlı olarak değerlendirilir.

Hangi yaklaşımım benimsenmesine karar vermek için bu yaklaşımların tam teferruatlı bilmek ve değerlendirmek gerekir.

**DBFirst Yaklaşımı**

EFCore ile işlem yapılacak veritabanı önceden var ise kesin olarak DBFirst yaklaşımını benimsenmemiz gerektiğini söyleyebiliriz.

DBFirst var olan veritabanını analiz ederek kod kısmında modelleyen bir yaklaşımdır.

* AVANTAJLARI

1. Hazır veritabanları hızlıca modellenebilir
2. Veritabanında süreçte olan değişikliklere göre hızlıca koda aktarmamızı sağlar
3. EFCore ile desteklenen tüm veritabanlarında kullanılabilir.

* DEZAVANTAJLARI

1. Kod veritabanına göre şekilleneceği için yönetim veritabanı tarafından sağlanır yani veritabanı bilgisi gerektirir.
2. Değişiklikler veritabanında olacağı için geliştirici tarafından sürekli kontrol davranışı sergilenmelidir.

* Hangi Durumlarda Tercih Edilmelidir?
* Önceden oluşturulmuş, veritabanı var olan uygulamalarda.
* Veritabanı yönetimine, geliştirme süreçlerine ve tasarımına dair herhangi bir kararın geliştiriciler tarafından verilmediği durumlarda tercih edilmelidir.

DBFirst, CodeFirst yaklaşımının tam tersidir veya CodeFirst,DBFirst yaklşaımının tam tersidir diyebiliriz.

**CodeFirst Yaklaşımı**

EFCore ile çalışma yapılacak olan veritabanı yok ise yani henüz oluşturulmamış ise bu veritabanını geliştirme aşamasında modelleyerek veritabanını sunucuda oluşturan (migration) yaklaşımdır.

Bu yaklaşımda veritabanı önce kodla tasarlanır, sonra db sunucusuna gönderilerek veritabanı oluşturulur.

* AVANTAJLARI

1. Kod üzerinde veritabanı modellememizi sağlar.
2. Veritabanına dokunmadan kod üzerinden veritabanındaki gerekli düzenlemelerin yapılmasını sağlar.
3. EFCore ile desteklenen tüm veritabanlarında kullanılabilir.
4. Koddaki ihtiyaca dönük veritabanı şekilleneceği için herhangi bir veritabanı bilgisine gerek duyulmamaktadır.
5. Değişiklikler kod kısmından yapılacağı için herhangi bir kontrol ihtiyacı söz konusu değildir.
6. Veritabanı modeli kod üzerinde yapıldığı için istenilen sunucuda anında ilgili modelin değerleri elde edilebilir.

* DEZAVANTAJLARI
  1. Üretilecek veritabanının tasarımı ve stratejisi geliştirici sorumluluğundadır.
* Hangi Durumlarda Tercih Edilmelidir
  1. Veritabanı bilgisine ihtiyaç duyulamayan
  2. Veritabanı tasarımının geliştirici tarafından üstlenilebileceği
  3. Veritabanı yönetiminin kod üzerinden sağlanacağı durumlarda tercih edilebilir.

**BAŞLIK**

* Bir ORM aracının veritabanını temsil edebilmesi için o veritabanındaki tabloların, tablolar içerisindeki kolon ve nesnelerin modellenmesi gerekmektedir.
* Bu modelleme class’lar üzerinden gerçekleştirilir.

VERİTABANI CLASSI: DbContext

* + EFCore da veritabanını temsil eden sınıf DBContext olarak nitelendirilmektedir.
  + Bir sınıfın veritabanına karşılık gelen sınıf olabilmesi için Microsoft.EntityFrameworkCore namespace’i altındaki DbContext sınıfından türemesi gerekmektedir.
  + Bakkal adında bir veritabanımız olsun ismi şu şekilde olacaktır: BakkalDbContext.

DbContext aşağıdaki faaliyetlerden sorumludur;

1. Konfigurasyon: Veritabanı bağlantısı, model yapılanmaları ve veritabanı nesnesi ile tablo nesneleri arasındaki ilişkileri sağlar
2. Sorgulama: Kod tarafında gerçekleştirilen sorgulama adımlarını SQL sorgusuna dönüştürür ve veritabanına gönderir.
3. Change Tracking: Sorgulama neticesinde veriler üzerindeki değişiklikleri takip eder
4. Veri Kalıcılığı: Verilerin kaydedilmesi, silinmesi veya güncellenmesi gibi operasyonları gerçekleştirir.
5. Caching

TABLO CLASSI: Entity

* + EFCore da tabloları temsil eden sınıflar entity olarak nitelendirilir.
  + Herhangi bir olguyu/nesneyi modelleyen sınıfa entity denir.
  + Tüm entity sınıfları DbContext sınıfı içerisine, DbSet olarak eklenmelidir. Böylece veritabanı sınıfı ile entity sınıfı arasında bir ilişki kurulacak ve EFCore, veritabanındaki tabloları hangi sınıflarla temsil edildiği bu ilişki üzerinden anlayacaktır.
  + DbSet<T> şeklinde DbContext içerisinde tanımlanır. Buradaki T tabloyu temsil edecek olan modeli içermelidir.
  + Veritabanında tabloyu temsil edecek olan property çoğul isimde olurken tablo içerisindeki model tekil isimde olacaktır. Yani Customers adlı bir tabloyu Customer adlı bir modelle temsil edeceğiz diyebiliriz. Dolayısıyla Customers içerisinde birden fazla Customer barındırabilir.
  + EFCore’da bir tabloyu temsil eden sınıf içerisindeki propertyler o tabloya ait kolonları temsil eder.
  + Veritabanındaki veriler entitylerin instancelarına karşılık gelmektedir.

**DbFirst Pratik**

İlgili veritabanının kod kısmında oluşturulması gerekmektedir. Bu işlemi PMC veya Dotnet CLI aracılığı ile iki farklı şekilde yürütebiliriz.

Connection strings: [https://www.connectionstrings.com]

PMC ile 🡺

Scaffold-DbContext ‘ConnectionString’ Microsoft.EntityFrameworkCore –[Provider]

ConnectionString:Server=myServerAddress;Database=myDataBase;User Id= myUsername; Password=myPassword;

Yüklenmesi gereken kütüphaneler:

* Microsoft.EntityFrameworkCore.Tools
* Database Provider

[İlgili databaselerin provideslerına alttaki linkenten erişebilirsiniz

<https://learn.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli>]





İlgili talimatları PMCden verdiğimizde veritabanımızın sınıfları ve Contextimiz oluşturulmuş olacaktır.

Dotnet CLI ile 🡺

dotnet ef dbcontext scaffold ‘ConnectionString’ [Provider]

Yüklenmesi gereken kütüphaneler:

* Microsoft.EntityFrameworkCore.Design
* Database Provider

cmd üzerinden:

dotnet ef dbcontext scaffold Server=(LocalDb)\HcSqlServer; Database= Northwind; Microsoft.EntityFrameworkCore.SqlServer

Talimatı verildiğinde veritabanımızın sınıfları ve Contextimiz oluşturulmuş olacaktır.

Varsayılan olarak databasedaki bütün tablolar modellenir. Ancak biz istenilen tablolaların modellenmesini istiyorsak aşağıdaki talimatları vermemiz gerekmekte:

Scaffold-DbContext ‘CS’[Provider] -Tables TableName1, TableName2…

dotnet ef dbcontext scaffold ‘CS’[Provider] --table TableName,--table TableName2..

TEKRAR TEKRAR YAZMAMAK ADINA “Scaffold-DbContext ‘CS’[Provider]” ve “dotnet ef dbcontext scaffold ‘CS’[Provider]” talimatlarını “Directive” olarak kısalttım

Eğerki oluşturulan context nesnesini adını değiştirmek istiyorsak;

“Directive” -Context ContextName (PMC)

“Directive” --context ContextName (CLI)

Path ve Namespace Belirtme

Entityler ve DbContext sınıfı varsayılan olarak projenin varsayılan namespaceni kullanır eğer ki bunlara müdahale etmek istiyorsanız aşağıdaki gibi talimat verebiliriz:

PATH:

“Directive” –ContextDir [Context Dosya Adı] –OutputDir [Model Dosya Adı] (PMC)

“Directive” –-[^^] Data –-[^^] Models (CLI)

[] içine yazılmamalı

NAMESPACE:

“Directive” -Namespace NameSpaceAdı -ContextNamespace NameSpaceAdı

“Directive” --namespace NameSpaceAdı --context-namespace NameSpaceAdı

Model Güncelleme

Eğer veritabanında kod yazma aşamasında bir güncelleme olurda veritabanı güncellenirse modellerimizi güncellememiz gerekebilir

Haliyle “Directive” kullanıldığında hata alınacaktır.

Bunun için “Directive” -Force ile güncelleyebiliriz

Modellerin Özelleştirilmesi

Modeller özelleştirildiğinde örneğin Order classına bir değer daha eklediğimizde -force komutu kullanıldığında ilgili değişiklik ezilecek yani silinecektir. Bu sorunu yaşamamak adına ilgili değişiklikleri yada özelleştirmeleri partial classlar altında yapmak en doğrusudur. Aksi halde özelleştirmeler ezilecektir.







Haliyle buradaki propertyimiz merkezi bir yapılanmada olabilir ve bizi mağduriyete sürüklememesi açısından;

HATIRLATMA AYNI İSİMDE AYNI NAMESPACE ALTINDA BİRDEN FAZLA PARTIAL CLASS TUTULABİLİR



Entitylerin altında aynı namespace altında aynı isimde classımızı oluşturup propertymizi tanımlıyoruz





Haliyle force yapıldığında ilgili property ezilmeyecektir.