Mi az Entity Framework? Mikor használjuk, mikor nem? Mi a DbContext? Milyen három modellezési lehetőségünk van adatmodell készítésére, és milyen két lehetőségünk van a modell konfigurációjára? Hogy néz ki egy entitás, hogyan készítünk kapcsolatokat közöttük? Hogyan készítünk lekérdezéseket, hogyan módosítunk adatokat EF Core-ban

DIA 5 4 oldal

Az Entity Framework egy .NET alapú ORM keretrendszer, ami a .NET Framework eredeti objektum-relációs leképező keretrendszere. Ezen kívül még a Entity Framework Core van, ami nyílt forráskódú és alapkoncepciójában megegyezik a simával csak modernizált és vannak kisebb nagyobb eltérések az eredeti EF-hez képest. Számozásban úgy kell nézni őket, hogy a EF Core 6 a .NET 6 keretrendszerrel kompatibilis.

DIA 5 6 oldal 9 oldal

Ezt akkor használjuk, ha a célunk a fejlesztési időben az adatbázis objektumok támogatása és a nekik megfelelő OO entitásmodell kialakításának összerendelése. Illetve a futási időben C# műveletek adatbázis műveletekre fordítsa le a framework, így ne kelljen adatbázisnyelvű kódot írni**. Akkor nem használjuk ezt, ha adatbáziskezelést alacsonyabb szinten szeretnénk kezelni**, **vagy mikro ORM-ek szeretnénk kialakítani**, aminek jellemzője, hogy kevesebb szolgáltatást nyújt, kevesebb fekete mágia van benne, egyszerűbben használatba vehető, illetve kisebb az overheadje így jobb a teljesítménye. Illetve az is lehetséges ok amiért nem választjuk ezt, hogy lehet a fejlesztők körében más ORM használat van aktívan például a NHibernate.

Dia 5 21 oldal

A DbContext egy olyan dolog, amiben az entitásmodellünket fogjuk össze. Ez reprezentálja az adatbázis a programunk számára. Ehhez viszont már kell a EF NuGet csomag függőség mivel magától nem importálja minden projektbe. Általában ebben a fájlban van egy OnConfiguring konstruktor, ahol inicializálunk mindent, ami kell, például kapcsolódási adatok kezelése naplózás. Illetve adunk táblánkként egy DbSet parancsot, amiben megadjuk az entitást is, hogy milyen típusú dolgok vannak benne. Illetve lehetséges itt megadni mapping kódot az OnModelCreating függvényben.

Adatmodellezés:

1. **Code-First elsőnek a kódot írjuk meg és az alapján generáltatjuk az adatbázis.**
2. **Database-First amikor az adatbázist csináljuk meg elsőnek és a kódot generáltatjuk**
3. **Model-First amikor a modelleket és kapcsolatokat határozzuk meg és automatikusan generáljuk az entitásokat és az adatbázismodellt.**

A modell tudjuk még úgy konfigurálni, hogy attribútumokat ahol testre szabhatjuk az entitások egyes tagjait. Például a StringLength taggal ellátva megmondhatjuk milyen hosszú lehet egy bizonyos string.

Az attributumok konfigurációjánál ha nem automatikus megoldást szeretnénk akkor tudunk attributumokat használni vagy fluent api-t. Attributumoknál a attributum felé tesszük a kívánt változtatást, mint [Key], [Column(„Name”)]

De fluent api-t is használhatjuk, amivel teljeskörű konfigurációt kapunk az attribútumos megoldás helyett. Itt az például OnModelCreating függvényben megmondhatjuk, hogy .HasMaxLength(xx) és ezzel oldjuk meg az előbbi problémát.

Kapocslatokból lehet egy kapcsolat egy-egy egy-több és több-több -es kapcsolat.

Az egy egyes kapcsolatnál mindkét elemnek egy idegen eleme lesz a másiktól illetve egy idegen kulcsa és a konfugurációban megmondjuk, hogy hasOne withOne és hasForeignKey.

Az egy többesnél csak az egyiknél van az elöző megoldás a másiknál egy lista lesz az idegen elemből. Itt az fogjuk konfigurálni az egyes oldalról hogy hasOne WithMany és adunk neki is egy hasForeignKey-t

A több többesnél már jóval több megoldás van de a legegyszerűbb ha mindkét entitásnak van egy listája a másikból és hasMany withMany használunk és utána egy usingEntity-ben taglaljuk az új kapcsoló tábla mikkéntjét. Például nevét és hogy milyen idegen kulcsok és tagok lesznek benne, esetleg plussz property-k. Viszont ha túl sok property-t szeretnénk konfigurálni akkor érdemesebb csinálni egy külön entitást ami a kapcsoló tábla lesz.

Lekérdezéseket a DbSet<entitás> ból tudunk kérni LINQ segítségével. Például van egy valamilyen Context amiben vannak tanulók és szeretnénk lekérni a Szabolcs nevű tanulókat. Akkor azt írjuk, hogy

var result = ctx.Students.Where(r => r.FirstName == „Szabolcs”).toList();

Ez persze nem aszinkron de van a listás megoldásnak aszinkron verziója is de akkor egy CancelationTokent kell megadni neki.

Modisítást úgy végzünk, hogy csinálunk egy lekérdezést abban modósítunk valamit majd azt vissza töltjük a kontextusba és mentjük.

ctx.Students.Remove(student);

ctx.SaveChanges();