Entity Framework Code First

Entity Framework je ORM framework, který usnadňuje vývoj aplikací využívajících přístup k relační databázi (byť verze 7 již počítá i s NoSQL databázemi). První verze tohoto frameworku byla představena v roce 2008 a byla označena jako Entity Framework 3.5 na základě tehdejší aktuální verze .NET frameworku 3.5 SP 1. Číslování Entity Frameworku se dramaticky změnilo až s příchodem [páté verze](http://weblogs.asp.net/scottgu/announcing-entity-framework-code-first-ctp5-release), která byla distribuována odděleně od .NET frameworku formou [NuGet](https://visualstudiogallery.msdn.microsoft.com/27077b70-9dad-4c64-adcf-c7cf6bc9970c) balíčku.

Po dobu své historie Entity Framework nabídl [tři odlišné přístupy práce s modelem](http://www.entityframeworktutorial.net/choosing-development-approach-with-entity-framework.aspx). První verze, označována jako Database First umožnila vytvořit schéma na základě existujícího fyzického databázového modelu. Zcela opačný přístup nabídl Model First, který umožnil generovat SQL skripty a následně databázové schéma na základě vizuálního schématu.

Problém vizuálních schémat vývojáři pocítili v momentě, kdy potřebovali provádět paralelně změny v modelu. Protože je vizuální schéma reprezentováno sadou souborů, které verzovací nástroje nedokáží snadno mergovat, paralelní vývoj s využitím DB First a Model First nemusel být vždy snadný.

Koncept [Code First byl představen s verzí Entity Framework 4.1](http://blogs.msdn.com/b/adonet/archive/2011/03/02/ef-4-1-is-coming-dbcontext-api-amp-code-first-rtw.aspx) a umožnil generovat schéma databáze na základě čistých entit ([POCO](https://msdn.microsoft.com/library/dd456872(v=vs.100).aspx)). Na základě různých popisů entit a vztahů mezi nimi dokáže až do dnešního dne EF nejen generovat schéma databáze ale zároveň provádět validaci, aniž by došlo k samotnému kontaktování databáze.

<https://www.miroslavholec.cz/blog/entity-framework-code-first-zarna-budoucnost>

Vztahy mezi entitami a validační pravidla jejich vlastností je v současné době možné popsat pomocí [datových anotací](https://msdn.microsoft.com/en-us/data/jj591583.aspx) nebo pomocí [Fluent API](https://msdn.microsoft.com/en-us/data/jj591617.aspx). Datové anotace jsou ve skutečnosti atributy, které se uvádí nad jednotlivými třídami nebo vlastnostmi. Technicky je každá anotace reprezentována specializací třídy [ValidationAttribute](https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.validationattribute(v=vs.110).aspx) a přetížením metody IsValid(). Fluent API je oproti tomu systémem extension metod, kterými je možné popsat vlastnosti entit nebo vztahy mezi nimi. Zatímco konfigurace pomocí Data Annotations je možná přímo nad názvy tříd a vlastností, Fluent API se inicializuje v metodě [OnModelCreating](https://msdn.microsoft.com/en-us/library/system.data.entity.dbcontext.onmodelcreating(v=vs.113).aspx) v třídě dědící tzv. [DbContext](https://msdn.microsoft.com/en-us/library/system.data.entity.dbcontext(v=vs.113).aspx). V praxi je možné Fluent api použít přímo v této metodě nebo s využitím tříd EntityTypeConfiguration a ComplexTypeConfiguration. Díky těmto třídám je možné model [popisovat zcela mimo kontext](http://daveaglick.com/posts/custom-entity-type-configurations-in-entity-framework-code-first-part-1) i mimo entity a docílit tak čistějšího návrhu. Data Annotations a samotné Fluent API se vzájemně nevylučují, byť mícháním obou metod dochází k znepřehlednění vztahů a omezení.

Pro práci se samotnou databází slouží její abstrakce, kterou je třída DbContext. Aby vývojář mohl pracovat s EF, musí tuto třídu dědit a nově vzniklé specializované třídě přidat řadu kolekcí, které reprezentují tabulky v databázi. Tyto kolekce jsou typu [DbSet<T>](https://msdn.microsoft.com/en-us/library/system.data.entity.dbset(v=vs.113).aspx) a jejich typem jsou již zmíněné POCO entity. Nad každým takovým DbSetem je posléze možné provádět typické CRUD [operace s využitím DbContext API](https://msdn.microsoft.com/en-us/data/gg192989.aspx) (tedy opět systémem extension metod) a jazyka LINQ.

Změny v databázi (čili v DbContextu) je nutné potvrdit zavoláním metody SaveChanges. Ta interně spouští systém sledování změn (tzv. [Change tracking](http://www.entityframeworktutorial.net/change-tracking-in-entity-framework.aspx)), který vyhledá změny v modelu a na základě těchto změn později EF vygeneruje SQL dotazy, které pošle v transakci proti databázi. Pokud taková transakce selže, změny nejsou provedeny a automaticky je proveden Rollback. Před samotným provedením změn EF spouští systém validace. Pro validaci entit slouží zmíněné datové anotace nebo omezení definovaná pomocí Fluent API. Jednotlivá validační pravidla nemusí být nutně takového charakteru, že by měla svůj ekvivalent na straně databáze. Entity tedy mohou mít řadu validačních pravidel, která na straně databáze neexistují a jsou ověřované na úrovni aplikace. V určitých případech není vyloučena i opačná situace, kdy některé omezení je definováno v databázi, ale v modelu chybí. Příkladem mohou být index a unique constraints, které mají v EF podporu teprve od verze 6.1. I v takovém případě selže celá transakce, byť až poté, co je zahájena komunikace s databázovým serverem.

Z [hlediska výkonnosti](https://www.google.cz/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwjA9eDq0MTJAhUHXBQKHb3TDLEQFggiMAA&url=https%3A%2F%2Fmsdn.microsoft.com%2Fen-us%2Fdata%2Fhh949853.aspx&usg=AFQjCNFp7CAQmDjsyOr23V-Bin8fymHohA&sig2=ucMml7sK910Ci6vOR7zNKw) je systém sledování změn velmi náročný a proto je možné jej vypínat buď nad množinami DbSet pomocí extension metody [AsNoTracking](http://www.c-sharpcorner.com/UploadFile/ff2f08/entity-framework-and-asnotracking/) nebo i globálně například v konstruktoru vlastního DbContextu. Vypnutím systému sledování změn nicméně vývojář přichází o možnosti validace. Výkonnostní rozdíl při zapnutém nebo vypnutém change trackingu je v desítkách procent.

<https://www.miroslavholec.cz/blog/entity-framework-a-optimalizace-dotazu>

EF umožňuje provádět standardní mapování běžných vztahů 1:N, M:N, 1:0/1 i dědičností typu [TPH](http://weblogs.asp.net/manavi/inheritance-mapping-strategies-with-entity-framework-code-first-ctp5-part-1-table-per-hierarchy-tph), [TPT](http://weblogs.asp.net/manavi/archive/2010/12/28/inheritance-mapping-strategies-with-entity-framework-code-first-ctp5-part-2-table-per-type-tpt.aspx) nebo [TPC](http://weblogs.asp.net/manavi/inheritance-mapping-strategies-with-entity-framework-code-first-ctp5-part-3-table-per-concrete-type-tpc-and-choosing-strategy-guidelines). Výchozím typem dědičnosti je TPH, která ukládá data v jediné tabulce. Každý řádek tabulky je rozlišován atributem Discriminator, který lze pomocí Fluent Api přejmenovat (včetně ukládaných hodnot). Typ TPC má horší podporu, protože bázová třída specializací je mapována vždy na unikátní tabulky. Z toho důvodu není možné používat jako primární klíč číslo s inkrementací ale buď řídit tvorbu ID ručně nebo použít tzv. [GUID](https://msdn.microsoft.com/en-us/library/dd354925.aspx) typ. Z hlediska mapování nakonec EF podporuje i [komplexní typy](https://visualstudiomagazine.com/articles/2014/04/01/making-complex-types-useful.aspx) a [enumy](https://msdn.microsoft.com/en-us/data/hh859576.aspx). Ke komplexním typům je vhodné zmínit možnost pojmenování sloupců dle potřeby, byť v případě použití komplexního typu vícekrát v rámci jedné entity tuto možnost vylučuje (konflikt názvu sloupců).

Konfigurace chování EF je možná pomocí řady nastavení, která se provádějí přímo v konstruktoru. Parametrický konstruktor třídy DbContext přijímá jako parametr připravené spojení nebo název connection stringu. Pokud vznikne instance specializované třídy vycházející z DbContext, provede se nejprve konstruktor a následně metoda OnModelCreating. Samotná reflexe modelu do databáze proběhne až při pokusu o prní kontakt z databází. K tomu dochází při prvním načítání dat nebo explicitním zavolání InitializeContext.

Proces inicializace je podřízen [zvolenému Initializeru](http://www.entityframeworktutorial.net/code-first/database-initialization-strategy-in-code-first.aspx). Tím výchozím je CreateDatabaseIfNotExists, který založí novou databázi, pokud ještě neexistuje. Pro účely ranné fáze vývoje se používají i další initializery označené jako DropCreateDatabase{…}. Každý z těchto inicializerů je možné dědit a implementovat vlastním kódem. Společnou metodou všech initializerů je Seed, která se spouští za předpokladu splněného pravidla pro inicializaci a uvnitř této metody se typicky definují výchozí data pro model.

V pokročilé fázi vývoje aplikací se používají databázové migrace. Ty mohou být automatické nebo manuální. Oba typy migrací používají speciální databázovou tabulku [MigrationsHistory](https://msdn.microsoft.com/en-us/data/dn456841.aspx), která ve speciálním sloupci Model uchovává XML schéma modelu a kterou je možné podle potřeb upravovat. Pokud EF zjistí, že se schéma v cílové databázi liší od lokálního a toto schéma v cílové databázi je zastaralé, vyžaduje aktualizaci. Aktualizaci je pak možné provést z ručně vytvořených migrací přímo z Visual Studia. Proces aktualizace může usnadnit speciální inicializer MigrateDatabaseToLatestVersion, který provede aktualizaci databázi automaticky, pokud se verze modelů neshodují (a db verze modelu je starší oproti aktuálnímu modelu). Aktualizace proběhne buď z ručně generovaných migrací, nebo v případě automatických migrací přímo z kódu.

<https://www.miroslavholec.cz/blog/code-first-initializers-a-migrace-kompletni-prehled>

V neposlední řadě EF podoruje režim Reverse Engineering. V tomto režimu se pracuje s databází, která neobsahuje tabulku Migrations History a vývojář se zcela svépomocí stará o synchronizaci databázového modelu vůči modelu aplikace. Tento režim lze s úspěchem využít v případech, kdy již existuje databáze a tato databáze není pod kontrolou vývojářského týmu.