Entity Framework Core 1

Albert István

ialbert@aut.bme.hu

Q.B. 221, 1662

BME AUT

Automatizálási és
Alkalmazott
Informatikai Tanszék

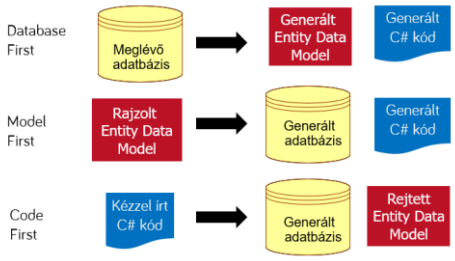
Tartalom

Bevezetés

- Adat != Objektum
  - > ORM

- LINQ to SQL és társai
  - > Közvetlen leképzés az adattáblák és az osztályok között
    - tábla ⇔ osztály
  - > Egyszerűbb adatmanipuláció ☺
  - > *Az adatmodell továbbra is erősen kihat az üzleti logika szerkezetére* ☹

Leképezési módszerek



Objektum gráf beszúrása

- Teljes objektum gráf beszúrása
  - > Az objektumok a megfelelő sorrendben kerülnek az adatbázisba: gyerekek először
  - > A külső kulcsok így jól állítódnak
- Egy vagy több kapcsolódó entitás esetén is működik

Entitás törlése

- A DbSet . Remove metódusával
  - > Ha az entitás a DbContextben Added állapotú: egyszerűen törlődik a DbContextből
  - > Ha az entitás a DbContextben NEM Added állapotú: a SaveChanges metódus kiadja a Delete parancsot

```
using (var db = new BloggingContext())
{
  var blog = db.Blogs.First(); // unchanged állapotú
  db.Blogs.Remove(blog);
  db.SaveChanges();
}
```


Módosítás

- A DbContextben lévő objektumok változásait automatikusan észleli
- A SaveChanges adja ki a parancsokat az adatbázis szervernek
  - > Egyetlen nagy tranzakciót használ

```
using (var db = new BloggingContext())
{
  var blog = db.Blogs.First();
  blog.Url = "http://sample.com/blog";
  db.SaveChanges();
}
```


Minta a leképezésre

- Ha a hierarchiában lévő osztályok explicit megjelennek a model deklarációban, akkor az EF kezeli a hierarchiát
- Külön DbSet-ként

```
public DbSet<Blog> Blogs { get; set; }
public DbSet<RssBlog> RssBlogs { get; set; }
```

- API-val megadva

```
protected override void OnModelCreating(ModelBuilder mb)
{
  mb.Entity<RssBlog>();
}
```


Egyszerű lekérdezések

- Bővítő metódusokkal vagy LINQ-vel
- Lekérdezés komponálás: a meglévő query tovább írható, bővíthető!

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Where(b => b.Url.Contains("dotnet"))
        .ToList();
}
```

Connection string

- Klasszikus .NET Framework alkalmazásokban
  - > app.config, web.config
- .NET Core: konfig fájl, env var, user secret store, …

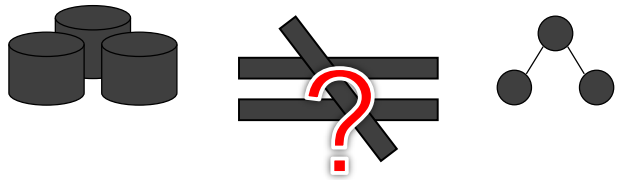
```
public void ConfigureServices(IServiceCollection services) {
    services.AddDbContext<BloggingContext>(
      options => options.UseSqlServer(
      Configuration.GetConnectionString("BloggingDatabase")));
}
```

Kérdések?

Albert István
ialbert@aut.bme.hu

Bevezetés

- Adat != Objektum
  - ORM



- LINQ to SQL és társai
  - Közvetlen leképzés az adattáblák és az osztályok között
    - tábla ⇔ osztály
  - Egyszerűbb adatmanipuláció ☺
  - *Az adatmodell továbbra is erősen kihat az üzleti logika szerkezetére* ☹

Az adatmodell erősen kihat az üzleti logikára

- Az adatbázissémák nem mindig ideálisak az éppen készülő alkalmazások számára
  - Meglévő sémákra kell épülniük az új alkalmazásoknak
  - Alul- vagy felülnormalizált sémák működési vagy teljesítményszempontok miatt
  - Az alkalmazás és az adatbázis is fejlődik az idővel
- Az adatbázisséma gyakran átitatja az egész alkalmazást
  - Minden alkalmazás próbál számára logikus nézeteket létrehozni
    - Tárolt eljárások, nézetek és (a leggyakrabban) ad hoc lekérdezések
  - A séma helyenként döntően visszahat az alkalmazás felépítésére

Entity Framework (EF)

- Testreszabható ORM rendszer
- Lehetővé teszi a ***logikai*** (*adatbázis*) és a ***fogalmi*** (*üzleti logika*) modellek szétválasztását és megfeleltetését
  - A fogalmi szint és a logikai szint közti különbség áthidalására lehetővé kell tenni a kettő közti automatikus leképzést
- „Függetleníti” az alkalmazásunkat az adatbázismotortól és az adatbázis sémától

Entitások

- A modell által leírt elemek az entitás típusok
- A modell az entitás típusokhoz tulajdonságokat rendel, de viselkedésüket nem definiálja
- Az entitások más entitásokhoz relációkkal kapcsolódhatnak

Adatbázis

- Az EF nem rendelkezik közvetlen ismeretekkel az adatbázismotorról
- Az adatbázismotor típusa (elvileg) nincs közvetlen hatással az EF működésére
- Az entitásokon végzett műveleteket egy külön almodul (provider) fordítja le az adatbázismotor műveleteire
- Támogatott szolgáltatók:
  - Microsoft SQL Server, CE
  - SQLite, Postgres, IBM Data Servers, MySQL
  - Oracle (Devart providerrel)
  - **InMemory adatbázis teszteléshez**

EF megvalósítás

- Linq to SQL: korai próbálkozás
- EF 1.0: 2008-ban, használhatatlanul kezdetleges
  - ObjectContext-et használ
- EF 4.0: 2011
  - DbContext, code first támogatás
- EF 6.0: 2013, open source
- EF Core 1.0: 2016, open source, cross platform
  - Még kezdetleges de alakul
- EF Core 2.0: 2017
  - .NET Standard 2.0

Entitás osztályok

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public List<Post> Posts { get; set; }
}
 public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

DbContext

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
```

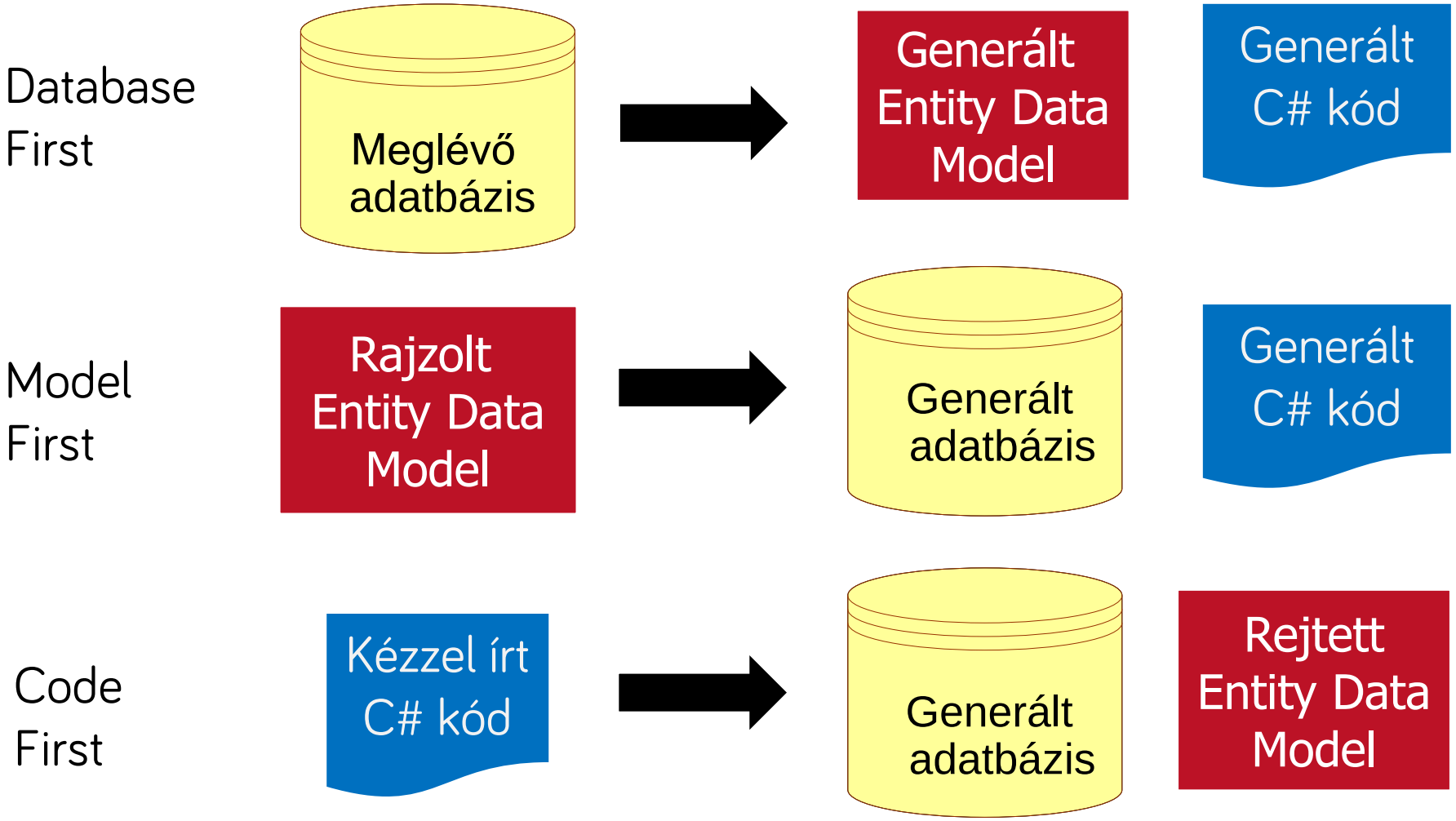
Egyszerű használat

```
using (var db = new BloggingContext())
{
        var blog = new Blog { Name = "..." };
        db.Blogs.Add(blog);
        db.SaveChanges();

        // Display all Blogs from the database
        var query = from b in db.Blogs orderby b.Name select b;

        foreach (var item in query)
            Console.WriteLine(item.Name);
}
```

Leképezési módszerek



Database first

- Van meglévő adatbázis
  - Az adatbázis elkészítése tetszőleges DB eszközzel
- Kód generálása adatbázis alapján
  - A navigáció neveit utána meg kell adni / át kell írni
  - További leképezés változtatások (például öröklés, stb)
- **Frissítés**
  - Az adatbázis módosul, tetszőleges eszközzel
  - Az adatmigráció az adatbázis módosításhoz tartozik
  - A kézi módosításokat megpróbálja megtartani
    - Provider függő, nem mindig sikerül ☹ ☹ ☹

Model first (egyelőre csak EF 6)

- Az EDMX designerben tervezed az entitás modellt
- A designerből generálható az adatbázist létrehozó szkript
- **Frissítés**
  - Az Entity Model-t módosítod
  - Generálod a teljes SQL szkriptet
  - Vagy eldobod a régi adatbázist és újat hozol létre
  - Vagy inkrementális szkriptet állítasz elő
  - Az adatmigráció kézzel történhet az inkrementális szkriptben

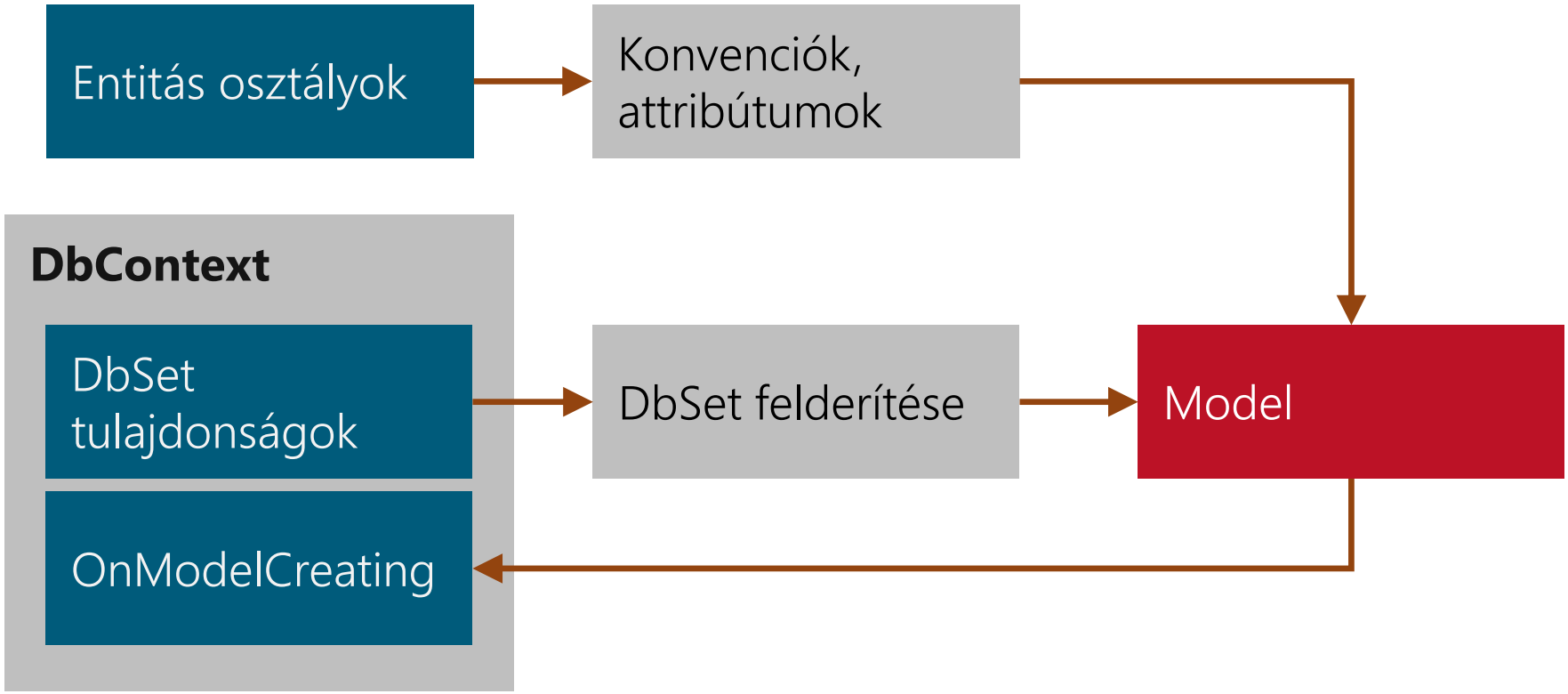
Code first

- Tisztán kód alapú megközelítés
  - A nomenklatúra inkább OO semmint EF
- Az adatbázis a kód alapján generálható
- **Migráció** is történhez kódból
  - Vagy törlöd az adatbázist
  - Vagy kivételt kapsz: az adatbázis nem felel meg a kódban leírt modellnek

Code First megközelítés

- Nincs dizájner, a kód határozza meg a leképezést!
- Konvenció
  - Id / OsztálynévID – ez az elsődleges kulcs
- Annotációk
  - …ComponentModel.DataAnnotations névtér
  - Kulcs, oszlopnév, szöveg hossz, stb.
- DbModelBuilder API
  - Programozottan
  - OnModelCreating felüldefiniálásával, ModelBuilder használatával

Model építés folyamata



Elsődleges kulcs

- Konvenció
  - Id
  - *típusnév*Id
- Annotáció
  - Key attribútum
- API

```
class Car
{
  public string Id { get; set; }
```

```
class Car
{
  public string CarId { get; set; }
```

```
class Car
{
  [Key]
  public string LicensePlate { get; set; }
```

```
protected override void OnModelCreating
                                (ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Car>() . HasKey(c => c.LicensePlate);
}
```

Generált értékek

- **Konvenció alapján csak a int vagy GUID elsődleges kulcs lesz automatikusan generált a szerver oldalon**

1. Csak új entitás létrehozásakor
2. Entitás létrehozásakor és módosításakor

- A provideren is múlik, hogy a kliensen az EF készít egy ideiglenes kulcsot vagy a szervert használja
  - > Ha már van kulcs beállítva, az EF megpróbálja beszúrni az adatbázisba

Generálás módja SQL Server esetén

- Az adatbázis provideren és a használt típuson múlik, például SQL Server esetén
  - GUID: automatikusan generálódik az adatbázisban
  - byte [ ] concurrency tokenként automatikusan rowversion lesz és automatikusan generálódik
  - DateTime: külön meg kell adni

Annotációk generáláshoz

```
public class Blog
{
  [DatabaseGenerated(DatabaseGeneratedOption.None)]
  public int BlogId { get; set; }
  public string Url { get; set; }

  [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
  public DateTime Inserted { get; set; }

  [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
  public DateTime LastUpdated { get; set; }
}
```

Kényszerek

```
public class Blog
{
  public int BlogId { get; set; }

  [Required]
  [MaxLength(500)]
  public string Url { get; set; }
}
```

Relációk konvenció alapon

```
public class Blog
{
  public int BlogId { get; set; }
  public string Url { get; set; }

  public List<Post> Posts { get; set; }
}

public class Post
{
  public int PostId { get; set; }
  public string Title { get; set; }
  public string Content { get; set; }

  public int BlogId { get; set; }    // külső kulcs, opcionális
  public Blog Blog { get; set; }
}
```

Több kapcsolat két entitás között

```
public class Post
{
  …
  public int AuthorUserId { get; set; }
  public User Author { get; set; }

  public int ContributorUserId { get; set; }
  public User Contributor { get; set; }
}

public class User
{
  …
  [InverseProperty("Author")]
  public List<Post> AuthoredPosts { get; set; }

  [InverseProperty("Contributor")]
  public List<Post> ContributedToPosts { get; set; }
}
```

Több-többes kapcsolatok

- Még nem támogatottak EF Core-ban
- Explicit létre kell hozni a kapcsoló táblát és egy-többes kapcsolattal megadni

Shadow properties 1

- Olyan tulajdonságok, amik nem jelennek meg az entitás osztályokon propertyként
- Csak a ChangeTrackeren keresztül lehet őket kezelni
  - Tipikusan infrastruktúrához tartozó tulajdonságok, funkciók

```
context.Entry(myBlog).Property("LastUpdated").CurrentValue =
DateTime.Now;
```

- Lekérdezésekben felhasználhatóak

```
var blogs = context.Blogs
    .OrderBy(b => EF.Property<DateTime>(b, "LastUpdated"));
```

Shadow properties 2

- Automatikusan létrejönnek, ha egy kapcsolat csak referenciaként jelenik meg de nincs külső kulcs az osztályon
- Fluent API-val deklarálhatók

```
class MyContext : DbContext
{
  public DbSet<Blog> Blogs { get; set; }

  protected override void OnModelCreating(ModelBuilder mb)
  {
    mb.Entity<Blog>()
                    .Property<DateTime>("LastUpdated");
} }
```

Index

- Alapértelmezetten minden külső kulcsként használt mezőhöz létrejön index
- Csak API-val lehet megadni
  - Unique / összetett indexet is meg lehet adni

```
protected override void OnModelCreating(ModelBuilder mb)
{
  mb.Entity<Blog>()
    .HasIndex(b => b.Url)
    .IsUnique();

  mb.Entity<Person>()
    .HasIndex(p => new { p.FirstName, p.LastName });
}
```

Nem leképezett típusok, kapcsolatok

- NotMapped attribútumot lehet használni típuson és propertyn is

```
public class Blog
{
  public int BlogId { get; set; }
  public string Url { get; set; }

  [NotMapped]
  public DateTime LoadedFromDatabase { get; set; }
}
```

Mezők használata propertyk helyett

- Ilyenkor nem a property settert használja az EF
- Konvenció alapján

```
public class Blog
{
  private string _url;

  public string Url  {
    get { return _url; }
    set { _url = value; }
} }
```

Leképezés relációs adatbázisra

- Táblák, mezők, típusok

```
[Table("blogs", Schema = "blogging")]
public class Blog
{
  [Column("blog_id")]
  public int BlogId { get; set; }
  [Column(TypeName = "varchar(200)")]
  public string Url { get; set; }
}
```

- Szekvencia, alapértelmezett érték, elsődleges/külső kulcs, index és reláció nevek API-val

Számított mezők

- Az adatbázis szerver számolja

```
protected override void OnModelCreating(ModelBuilder mb)
{
  mb.Entity<Person>()
              .Property(p => p.DisplayName)
              .HasComputedColumnSql
                    ("[LastName] + ', ' + [FirstName]");
}
```

Objektum gráf beszúrása

- Teljes objektum gráf beszúrása
  - Az objektumok a megfelelő sorrendben kerülnek az adatbázisba: gyerekek először
  - A külső kulcsok így jól állítódnak
- Egy vagy több kapcsolódó entitás esetén is működik

Példa objektum gráf felvételére

```
var db = new BlogDbContext();
var post = new Post() {
  Title = "New Post Title", Date = DateTime.Now,
  Body = "This post have comments and tags",
  User = db.Users.First(),
  Comments = new Comment[] {
    new Comment() { Text = "Comment 1", Date = DateTime.Now },
    new Comment() { Text = "Comment 2", Date = DateTime.Now,
                    User = db.Users.First() } },
                    Tags = db.Tags.Take(3).ToList()
};
db.Posts.Add(post);
db.SaveChanges();
```

Meglévő szülőhöz új entitás

```
using (var context = new BloggingContext())
{
  var blog = context.Blogs.First();
  var post = new Post { Title = "Intro to EF Core" };

  blog.Posts.Add( post );
  context.SaveChanges();
}
```

- A DbContext tud a blog objektumról a lekérdezés miatt
- Az Add metódus csatolja az új Post entitást a DbContexthez

Meglévő gyerekhez új szülő entitás

```
using (var context = new BloggingContext())
{
  var blog = new Blog { Url = "http://blogs.msdn.com/" };
  var post = context.Posts.First();

  blog.Posts.Add(post);
  context.SaveChanges();
}
```

- A DbContext tud Post objektumról
- A blog új, az Add metódus módosítja a Post szülő Blog tulajdonságát
- A post entitáson keresztül a blog entitás is a DbContext tudomására jut

Entitás törlése

- A DbSet . Remove metódusával
  - Ha az entitás a DbContextben Added állapotú: egyszerűen törlődik a DbContextből
  - Ha az entitás a DbContextben NEM Added állapotú: a SaveChanges metódus kiadja a Delete parancsot

```
using (var db = new BloggingContext())
{
  var blog = db.Blogs.First(); // unchanged állapotú
  db.Blogs.Remove(blog);
  db.SaveChanges();
}
```

Kaszkád törlés

- Adatbázis séma tulajdonságok
  - Az entitás külső kulcsa lehet-e NULL, azaz létezhet-e a „szülője” nélkül
  - Egy szülő törlésével az összes gyerek objektum automatikusan törlődik
- Az EF Core négy viselkedést különböztet meg:
  - **Cascade**: a függő entitások törlődnek, alapértelmezett, ha kötelezően kitöltendő a kapcsolat
  - **SetNull**: a függő entitásokon a külső kulcs NULL lesz
  - **Restrict**: a függő entitások nem változnak
  - **ClientSetNull:** a függő entitásokon a külső kulcs NULL lesz *még a memóriában (*ha opcionálisan kitöltendő a kapcsolat)

Kaszkád működés

- A kaszkád törlés a DbContext csak az általa ismert (a ChangeTrackerben szereplő) objektumokra alkalmazza
- Egyébként a kaszkád törlést az adatbázis végzi
- Ezt az adatbázison kell beállítani, ha az EF hozza létre az adatbázist, akkor azt beállítja a modellnek megfelelően

Kaszkád kapcsolat beállítása

```
class MyContext : DbContext
{
  public DbSet<Blog> Blogs { get; set; }
  public DbSet<Post> Posts { get; set; }

  protected override void
              OnModelCreating(ModelBuilder modelBuilder)
  {
    modelBuilder.Entity<Post>()
                .HasOne(p => p.Blog)
                .WithMany(b => b.Posts)
                .OnDelete(DeleteBehavior.Cascade);
  }
}
```

Törlés 1

```
using (var db = new BloggingContext())
{
  var blog = db.Blogs.Include(b => b.Posts).First();
  db.Remove(blog);
  db.SaveChanges();
}
```

- Minden már betöltött entitást explicit módon töröl

```
DELETE FROM [Post]
WHERE [PostId] = @p0;
DELETE FROM [Post]
WHERE [PostId] = @p1;
DELETE FROM [Blog]
WHERE [BlogId] = @p2;
```

Törlés 2

```
using (var db = new BloggingContext())
{
  var blog = db.Blogs.First();
  db.Remove(blog);
  db.SaveChanges();
}
```

- Az adatbázisra bízza a kapcsolódó entitások törlését

```
DELETE FROM [Blog]
WHERE [BlogId] = @p2;
```

Kapcsolat törlése

- Külső kulcs NULL értékre állítása
  - Ezt meg kell engednie az adatbázis sémának!
- A navigációs propertyket nullra állítjuk vagy a szülő gyűjteményből töröljük az entitást
  - Ha a kapcsolat opcionális, akkor NULL kerül az adatbázisba a gyerek entitáson
  - Ha a kapcsolat kötelező (nem lehet NULL az adatbázisban) és …
    - Kaszkád a reláció a szülővel, akkor a gyerek törlődik!
    - Nem kaszkád a reláció: kivételt kapunk!

Módosítás

- A DbContextben lévő objektumok változásait automatikusan észleli
- A SaveChanges adja ki a parancsokat az adatbázis szervernek
  - Egyetlen nagy tranzakciót használ

```
using (var db = new BloggingContext())
{
  var blog = db.Blogs.First();
  blog.Url = "http://sample.com/blog";
  db.SaveChanges();
}
```

Művelet objektum gráfokon

- A DbSet metódusai a teljes csatolt objektumgráf összes, eddig ismeretlen entitására beállítja az entitás állapotát a műveletnek megfelelően
  - **Add**: minden eddig ismeretlen entitás új lesz
  - **Attach**: minden entitás Unchanged, kivéve, ha annak szerver oldalon generált az azonosítója és még nincs kitöltve, ilyenkor új (Added) állaptú
  - **Update**: mint az Attach, csak módosított az állapot
  - **Remove**: mint az Attach, de csak a gyökér objektum lesz törölt állapotú. A SaveChanges metódus felel a kaszkádosításért!

TrackGraph API

- DbContext . ChangeTracker . TrackGraph
- Bejárja a teljes átadott gráfot és elvégzi a kívánt műveletet az ismeretlen entitásokon
- A DbContext számára már ismert objektumokat kihagyja, hasonlóan a DbSet műveleteihez

```
context.ChangeTracker.TrackGraph(
  newEntity, e => e.Entry.State = EntityState.Added);
```

Művelet gyűjteményeken

- DbSet metódusai
  - AddRange, RemoveRange, UpdateRange, AttachRange
- Ugyanúgy működik, mintha egyes objektumokon hívnánk a megfelelő metódusokat

Generikus metódusok

- Az DbContext is rendelkezik a klasszikus metódusokkal, a típus alapján kideríti, melyik DbSet-et kell meghívnia
  - Heterogén gyűjteményen is működik

Művelet egyetlen objektumon

- Ha a gráf egyetlen objektumának állapotát az Entry bejegyzés állapot tulajdonságával tudjuk módosítani

```
DbContext.Entry(object).State = EntityState.Added;
```

- IsKeySet tulajdonság: van-e kulcs beállítva

Minta a leképezésre

- Ha a hierarchiában lévő osztályok explicit megjelennek a model deklarációban, akkor az EF kezeli a hierarchiát
- Külön DbSet-ként

```
public DbSet<Blog> Blogs { get; set; }
public DbSet<RssBlog> RssBlogs { get; set; }
```

- API-val megadva

```
protected override void OnModelCreating(ModelBuilder mb)
{
  mb.Entity<RssBlog>();
}
```

TPH: Table Per Hierarchy leképezés

- Egy teljes öröklési fa minden típusa ugyanabba a táblába kerül
  - Széles tábla, minden típus mezője benne van
- Lekérdezés
  - Minden entitástípusnál meg van adva egy feltétel, mely alapján a típus beazonosítható (pl.: discriminator = „Person”)
  - Egyetlen táblát kérdez le

Table Per Type leképezés

- Még nincs EF Core-ban
- Minden típushoz tartozik egy tábla
  - Csak a típushoz tartozó mezők vannak a táblában
    - Az ősosztály mezői nincsenek benne
- Lekérdezés
  - Több táblát kell lekérdezni: a származási hierarchia mentén minden szülőt
  - De az egyes táblák kisebbek

Table Per Concrete Type leképezés

- Még nincs EF Core-ban
- Minden típushoz tartozik egy tábla
  - A típushoz tartozó összes mező benne van
    - az ősosztály mezői is
- Lekérdezés
  - Egy konkrét típus lekérdezésénél csak egyetlen táblát kell lekérdezni
  - Ősosztályt lekérdezve az összes leszármazotthoz tartozó táblát végig kell kérdezni

Egyszerű lekérdezések

- Bővítő metódusokkal vagy LINQ-vel
- Lekérdezés komponálás: a meglévő query tovább írható, bővíthető!

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Where(b => b.Url.Contains("dotnet"))
        .ToList();
}
```

Meglévő entitások felhasználása

- A query mindig az adatbázison fut, nem használjat a DbContexten meglévő entitásokat
  - Kivéve Find - csak akkor fordul DB-hez, ha nincs meg DBContext-ben az entitás
- Ha már vannak a DbContextben korábban meglévő entitások, akkor automatikusan azokat adja vissza a query az eredményekor

Kapcsolódó entitások betöltése

- **Implicit**: ha már vannak kapcsolódó entitások a memóriában, az EF automatikusan összeköti az objektumokat
- **Előtöltés** (eager loading): a kapcsolódó entitások betöltése az első lekérdezéssel együtt
- **Explicit betöltés** (explicit loading): már memóriában lévő entitáshoz további kapcsolódó entitások betöltése
- **Késleltetett betöltés** (lazy loading): kapcsolódó entitások transzparens betöltése
  - NE HASZNÁLD!!!

Előtöltés

- DbSet Include metódusa

```
var blogs = context.Blogs
    .Include(blog => blog.Posts)
    .ToList();
```

- IncludeThen: lefúrás

```
var blogs = context.Blogs
    .Include(blog => blog.Posts)
        .ThenInclude(post => post.Author)
        .ThenInclude(author => author.Photo)
    .ToList();
```

Explicit betöltés

- Kapcsolódó entitások, entitás halmazok explicit betöltése

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);

    context.Entry(blog)
        .Collection(b => b.Posts)
        .Load();

    context.Entry(blog)
        .Reference(b => b.Owner)
        .Load();
}
```

Nem követett entitások 1.

- Több-rétegű alkalmazásokban az összes lekérdezés ilyen lehet
  - A lekérdezett entitások nem kerülnek bele a DbContextbe
  - Alapértelmezetten ki van kapcsolva

```
using (var context = new BloggingContext())
{
  var blogs = context.Blogs
    .AsNoTracking()
    .ToList();
}
```

Nem követett entitások 2.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Select(b =>
            new {
                Blog = b,          // tracked
                Posts = b.Posts.Count()
}); }
```

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Select(b =>
            new {   // there is no blog to track
                Id = b.BlogId,
                Url = b.Url
}); }
```

Közvetlen SQL lekérdezések

- Egyszerű lekérdezés

```
var blogs = context.Blogs
    .FromSql("SELECT * FROM dbo.Blogs")
    .ToList();
```

- Paraméterezve (NEM összefűzéssel), tárolt eljárás hívással

```
var user = "johndoe";

var blogs = context.Blogs
  .FromSql("EXECUTE dbo.GetMostPopularBlogsForUser {0}", user)
  .ToList();
```

Kevert kiértékelés

- Néha jól jöhet

```
var blogs = context.Blogs . Select(blog => new {
        Id = blog.BlogId, Url = StandardizeUrl( blog.Url ) });
```

- Az EF alapértelmezetten naplóz egy figyelmeztetést, ha kliens oldalon értékel ki valamit
- A figyelmeztetés dobhat kivételt is

```
protected override void OnConfiguring(… optionsBuilder)
{
  .UseSqlServer(@"…")
  .ConfigureWarnings(warnings =>
    warnings.Throw(RelationalEventId.QueryClientEvaluationWarning));
}
```

Connection string

- Klasszikus .NET Framework alkalmazásokban
  - app.config, web.config
- .NET Core: konfig fájl, env var, user secret store, …

```
public void ConfigureServices(IServiceCollection services) {
    services.AddDbContext<BloggingContext>(
       options => options.UseSqlServer(
       Configuration.GetConnectionString("BloggingDatabase")));
}
```

- UWP: SQLite

```
public class BloggingContext : DbContext {
  protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder) {
    optionsBuilder.UseSqlite("Filename=Blogging.db");
} }
```

Naplózás

- ILoggerProvider
  - Létrehozza a naplózó komponenst, kategóriától függően
  - Kategória: a komponens neve, ami épp naplóz
- ILogger
  - Elvégzi a naplózást

```
public void Log<TState>(LogLevel logLevel, EventId eventId,
TState state, Exception exception, Func<TState, Exception,
string> formatter)
{
  File.AppendAllText(@"C:\temp\log.txt",
                              formatter(state, exception));
  Console.WriteLine(formatter(state, exception));
}
```