



Haladó Fejlesztési Technikák



ÓBUDAI EGYETEM
NEUMANN JÁNOS INFORMATIKAI KAR

MODUL 4

Adatbáziskezelés célja

Adatbázis motorok

ORM rendszerek

Entity Framework Core

Code-First koncepció

Database-First koncepció

- **TXT/XML/JSON fájlokban adattárolás**

- Teljesen jól használható 1 felhasználós/1 szálú környezetben
- Teljesítmény szempontból nem a legjobb
- Nem biztosít:
 - Tranzakciókezelést
 - Redundáns tárolást
 - Elosztott tárolási lehetőséget

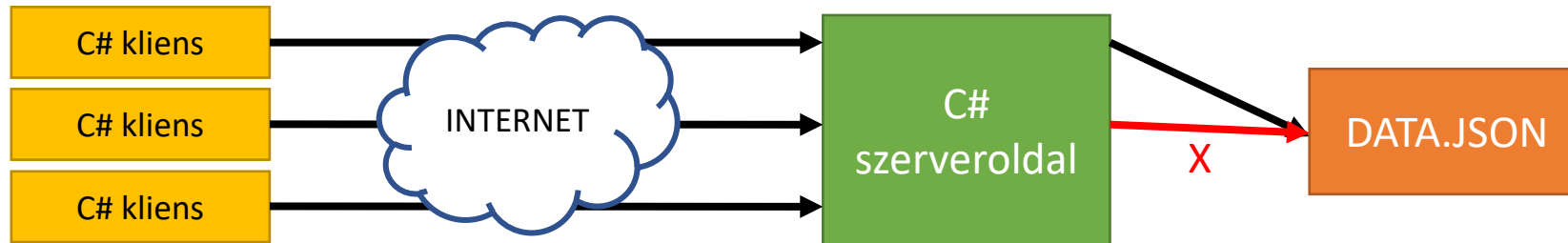
- **Mikor kell adatbázis?**

- Hogyha több szálon dolgozik az alkalmazásunk
- Ha az egyik szálon írjuk a fájlt, a másik szál nem fér hozzá → Exception
 - Megoldási ötlet: random ideig várakozás → torlódás, lassú válaszidő

Többszálú alkalmazás?!

4

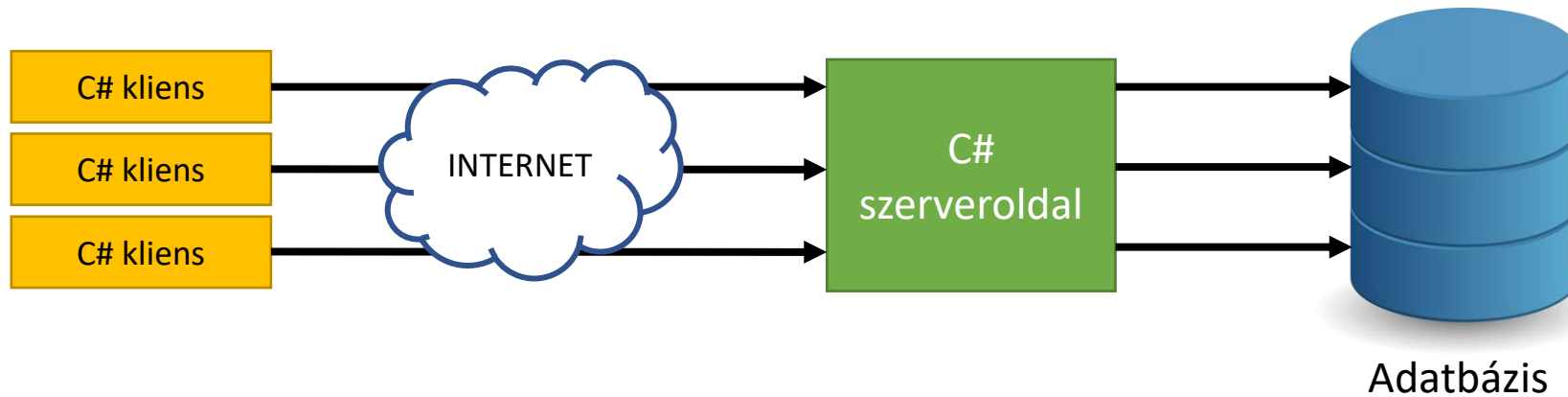
- Szerveroldali alkalmazás fejlesztés esetén → már ebben a félévben is
- Egy C# alkalmazás kérésekre figyel a HTTP 80-as porton
- Egy C# alkalmazás kéréseket küld a szerveralkalmazásnak → több példány is lehet
- A szerver alkalmazás ekkor minden kérést más szálon hajt végre párhuzamosan



Többszálú alkalmazás?!

5

- Szerveroldali alkalmazás fejlesztés esetén → már ebben a félévben is
- Egy C# alkalmazás kérésekre figyel a HTTP 80-as porton
- Egy C# alkalmazás kéréseket küld a szerveralkalmazásnak → több példány is lehet
- A szerver alkalmazás ekkor minden kérést más szálon hajt végre párhuzamosan



- **Egy szoftver, ami:**
 - Fel van telepítve a fejlesztő gépünkre/kiszolgálóra
 - Adott portszámon kérésekre válaszol (pl: MySql – 3306)
 - Elküldjük neki a tárolási igényeinket → megoldja valahogy
 - Elkérjük az adatokat → visszaadja valahogy
- **Gyakorlatilag az alkalmazásaink adattárolás („perzisztencia”) részét kiszervezzük egy programnak, ami:**
 - Párhuzamos kérések kezelésére képes
 - Redundánsan tárolja az adatokat
 - Tranzakcióként kezel minden kérést
 - Hiba esetén rollbackkel az előző jó állapotra

- **Kis-és középvállalatok által használt relációs adatbázis-kezelő szoftver**
 - **SQL Express:** kisebb változat, max. 10GB/adatbázis, max. 1 CPU, max 1GB RAM
 - **LocalDB:** Szerverszolgáltatás helyett egy azonos funkcionalitást nyújtó library, ami .mdf fájlba dolgozik
- **LocalDB használata**
 - **VS telepítéskor:** „Data Storage and Processing” workloadot telepíteni kell
 - Projekttel azonos könyvtárban .mdf és .ldf fájlok → átvihetők más gépre fájlként másolva
 - A kód ugyanaz, csupán a connection stringet kell átírni
 - A táblák minden futtatáskor visszaállnak az eredeti állapotra
 - Project / Add New Item / Service-based Database
- **InMemoryDatabase használata**
 - Valójában memóriában vannak az adatok tárolva
 - Teszteléshez jó
 - Féléves feladathoz ez lesz célszerű (MAC-en is működik)

- **ADO.NET: DbConnection technika**

- SQL utasítások küldése stringként
- Object tömb az eredmény → kasztolnunk kell saját adattípusra mindent stringből
- Connected mód

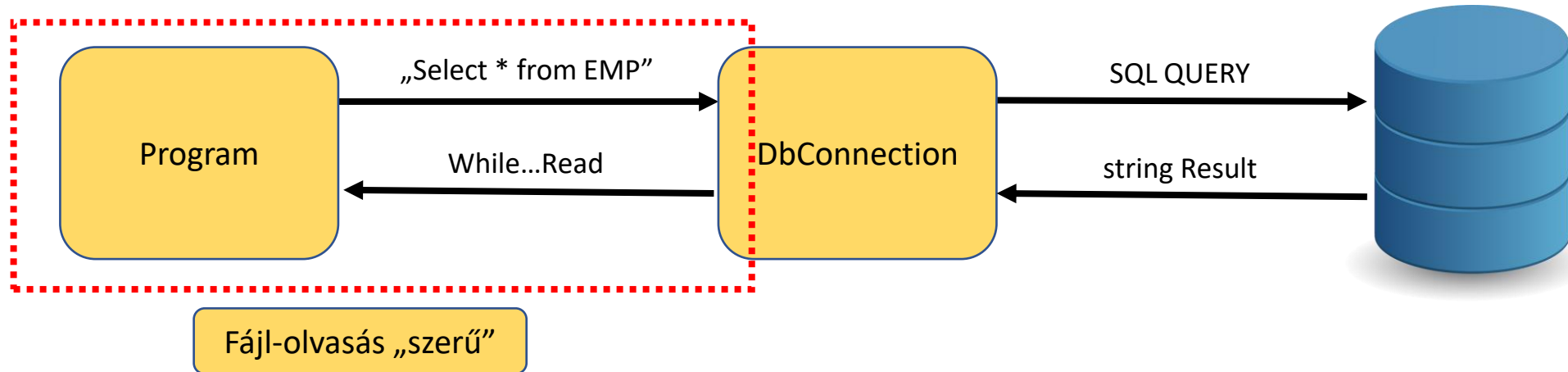
- **DataSet technika**

- GUI központú adatbáziskezelő réteg (Console-ban nem tudjuk használni)
- Már nem használt

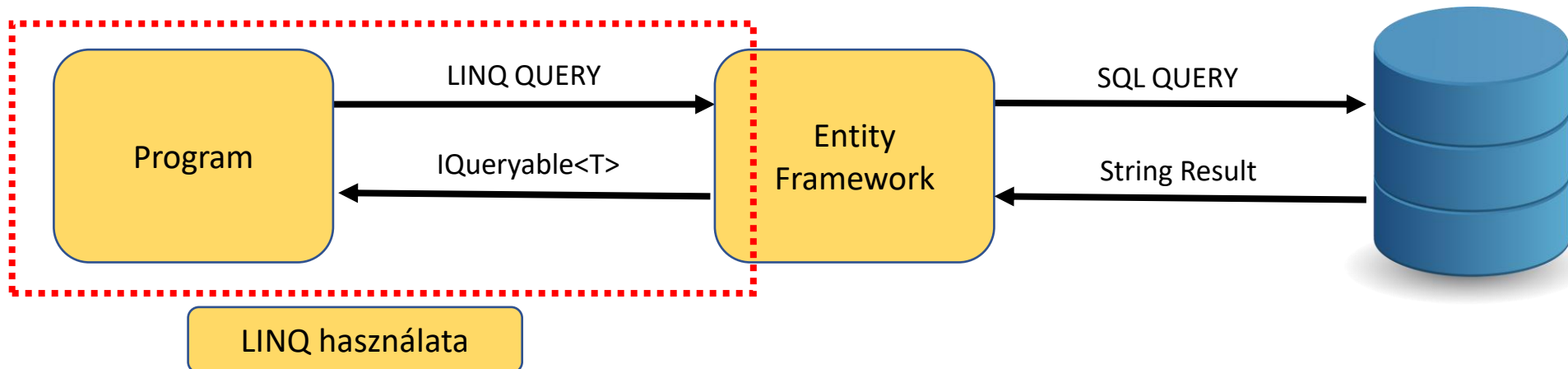
- **Entity Framework technika**

- SQL réteg fölé ORM (Object Relational Mapping) réteget helyez
- Táblákat mint objektumgyűjteményeket érjük el
- Connected/Disconnected mód

- **DbConnection technika (SQL utasítások)**



- **Entity Framework technika (ORM)**



- **Mindig minden** szűrést/rendezést/al-lekérdezés **bele KELL írni a QUERY-be**
- Egy adatbázisról feltételezzük, hogy NAGY (=akár több millió rekordos táblák)
- Elrettentő példa
 - Magyarország minden lakosának neve, címe, telefonszáma milyen méretű?
 - Adattípusok
 - Név → 100 hosszú string → 200 bájt UTF16-ban
 - Cím → 100 hosszú string → 200 bájt UTF16-ban
 - Telefon → 10 hosszú string → 20 bájt UTF16-ban
 - 10M lakos → 4,1 GB
 - Majdnem 2 perc letölteni hálózaton keresztül
 - Kb. 2 perc lefuttatni a szűrést i7-2600 processzoron
- A szűrést végezze el az adatbázis és csak azt a 4-5 rekordot kapjuk vissza, amire igaz a szűrési feltétel!

- Az adatbázis szerver is egy számítógép
- De az adatbázis szervereknek van saját megoldásuk a gyorsításra
 - Valamely mező szerint szétosztott működés (Sharding)
 - Eleve több kiszolgáló a read műveletekre (Replica Set)
 - Valamilyen kulcs mezők pl. név alapján hasítás
 - Sűrű lekérdezésekre cachelés
 - Gyors memória, SSD háttértár
 - Eleve memóriában tartott táblák
- **Egy szó mint száz:** minden feladatot az adatbázis-szerver végezzen el, mi csak az eredményeket jelenítsük meg
- **Megvalósítás:** Jól megírt LINQ lekérdezések kellenek, **ToList()** és egyébek tilosak LINQ-n belül

- „Kapcsolt” adatbázis-elérés (Connected Data-Access Architecture)
 - **Előny:** gyors, egyszerű
 - **Hátrány:** nehéz módosítani és technológiát/tárolási módszert váltani; kapcsolat elveszését kezelni kell
- A különböző adatbázis-szerverekhez különböző implementációk
- Közös őszosztályok a különféle feladatokhoz
 - Adatbázis-kapcsolat: **DbConnection**
 - SQL/RPC utasítás végrehajtása: **DbCommand**
 - Utasítás eredményének beolvasása: **DbDataReader**
- Specifikus utódosztályok a különféle adatbázis-szerverekhez
 - **SqlConnection** (MSSQL System.Data.SqlClient)
 - **MySqlConnection** (MySQL MySql.Data.MySqlClient)
 - **NpgsqlConnection** (PostgreSQL - Npgsql)
 - **OracleConnection** (Oracle System.Data.OracleClient)

- Connection String megszerzése

- MDF fájl projektbe helyezése / létrehozása (Add → New Item → Data → Service-based Database)
- View → Server Explorer → xyz.mdf jobb klikk → Property Editor ablak → Connection String
- **Server explorerben mindig jobb klikk → Close Connection / Detach database**

- Connection string létrehozása

```
string connStr = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\marvelmovies.mdf;Integrated Security=True";
```

- Példányosítás

```
SqlConnection conn = new SqlConnection(connStr);
```

- Megnyitás / bezárás

```
conn.Open();  
conn.Close();
```

- **SELECT**

```
SqlCommand cmd = new SqlCommand("select * from MOVIES", conn);  
SqlDataReader reader = cmd.ExecuteReader();  
while (reader.Read())  
{  
    Console.WriteLine(reader["Title"].ToString());  
}  
reader.Close();
```

utasítás

lefuttatás

Amíg van rekord, addig olvasás soronként

Adott rekord, adott eleme

Lezárás

- **INSERT**

```
SqlCommand cmd = new SqlCommand("insert into MOVIES (MovieId,Title,DirectorId)  
                                values (28,'Doctor Strange in the M.O.M',1)", conn);  
int affected = cmd.ExecuteNonQuery();  
Console.WriteLine("rows modified: " + affected);
```

- Hány mezője/oszlopa van egy adattáblának?

```
reader.FieldCount;
```

- Adott indexű mező nevének lekérése

```
reader.GetName(2);
```

- Adott nevű mezőhöz tartozó adat az aktuális rekordból (objectként)

```
object title = reader["Title"];
```

- Objectként adat az adott indexű oszlopból

```
reader.GetValue(4);
```

- Már megfelelő primitív típusként visszakérés (csak index alapján)

```
double rating = reader.GetDouble(5);
```

- **UPDATE**

```
SqlCommand cmd =  
    new SqlCommand("update MOVIES set Rating=9.9 where Title='Avengers: Endgame'", conn);  
int affected = cmd.ExecuteNonQuery();  
Console.WriteLine("rows modified: " + affected);
```

- **DELETE**

```
SqlCommand cmd = new SqlCommand("delete from ROLES where RoleName='Mordo'", conn);  
int affected = cmd.ExecuteNonQuery();  
Console.WriteLine("rows modified: " + affected)
```


- **SQL Injection ellen védekezni kell**

- Alapvető login kód

```
string uName = "yyy";  
string uPass = "xxx";  
string sql = $"SELECT * FROM users WHERE username='{uName}' AND userpass=sha2('{uPass}')";
```

- Mi történik ha valaki begépel a jelszava helyett ezt a UI-on/Konzolon?

- `x') OR 1=1 OR 1<>sha2('x`

- Így fog kiértékelődni az SQL query:

- `SELECT * FROM users WHERE username='Joe' AND userpass=sha2('x') OR 1=1 OR 1<>sha2('x')`

- **Hatás**

- Jelszó hiányában is be tud lépni a támadó, mert bekerült egy OR operátor a támadó által
 - Jó a jelszó VAGY 1 = 1 (ez mindig igaz lesz)

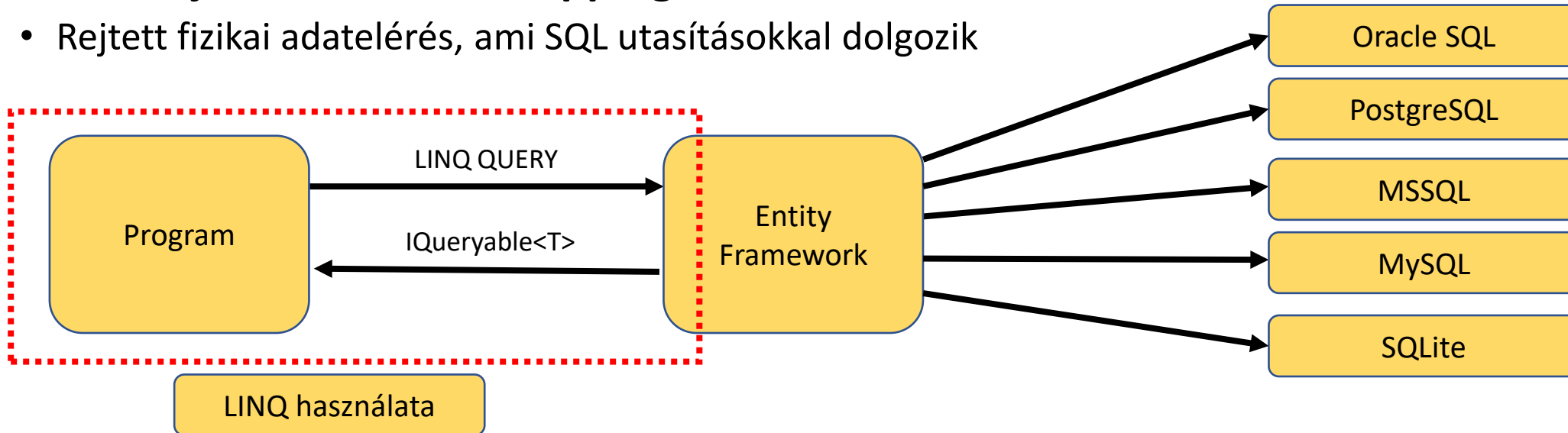
- **Üzleti logikai kódba kell helyezni az SQL kódot**
 - Ha áttérünk más tárolásra (pl. másfajta SQL vagy NoSQL), akkor át kell írni az üzleti logikai kódot
 - Üzleti logika soha ne függjön az adattárolás módjától
- Valamilyen réteg kell az SQL fölé, ami elrejtí az SQL kódot
 - Erősen típusos, SQL funkcionalitásával azonos réteget kellene az SQL réteg fölé helyezni
 - Ez pl. lehetne LINQ...

- Csatlakozzunk egy meglévő egy táblás adatbázishoz!
 - <https://nikprog.hu/samples/ado/marvel.mdf>
- Tábla felépítése
 - **MovieId**: azonosító (int)
 - **Title**: cím (string)
 - **Income**: bevétel (double)
 - **DirectorId**: rendező id (nem mutat sehova)
 - **Release**: megjelenés (DateTime)
 - **Rating**: értékelés (double)
- Jelenítsük meg a táblát konzolban!
- Szűrjünk be néhány sort!



- **ORM = Object Relational Mapping**

- Rejtett fizikai adatelérés, ami SQL utasításokkal dolgozik



- A főprogramunk számára csak objektumgyűjteményként látszik az adatbázis
 - Mintha végig egy `List<T>`-be dolgoznánk, amin tudunk LINQ-zni
 - De valójában egy adatbázis táblát módosítgatunk ezen keresztül
- **ORM más nyelvekben:** Java – Hibernate/JPA, PHP – Doctrine, Python - SQLAlchemy

- Előnyök

- Nincs dialektusfüggő SQL utasítás a kódban
- Nincs string formátumú SQL utasítás a kódban
- SQL injection ellen védett
- Lekérdezés eredménye típusos gyűjtemény

- Hátrányok

- Nehezebb konfiguráció
- Nagyobb memóriaigény
- Nagyobb CPU igény
- Bonyolultabb lekérdezések nem biztos, hogy támogatottak
- Nehéz optimalizáció

- Nuget: **Microsoft.EntityFrameworkCore** (verzió: 5.0.14)
 - Ez kell .NET 5-höz (de megjelent már a .NET 6-hoz a 6.0.2 verzió)
- Egyéb csomagok, amelyekre szükség lesz
 - Microsoft.EntityFrameworkCore.Proxies
 - Microsoft.EntityFrameworkCore.SqlServer
 - Microsoft.EntityFrameworkCore.Tools
- Célszerű feltelepíteni a **csproj** fájl szerkesztésével

```
<ItemGroup>
  <PackageReference Include="Microsoft.EntityFrameworkCore" Version="5.0.14" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Proxies" Version="5.0.14" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="5.0.14" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="5.0.14">
    <PrivateAssets>all</PrivateAssets>
    <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
  </PackageReference>
</ItemGroup>
```

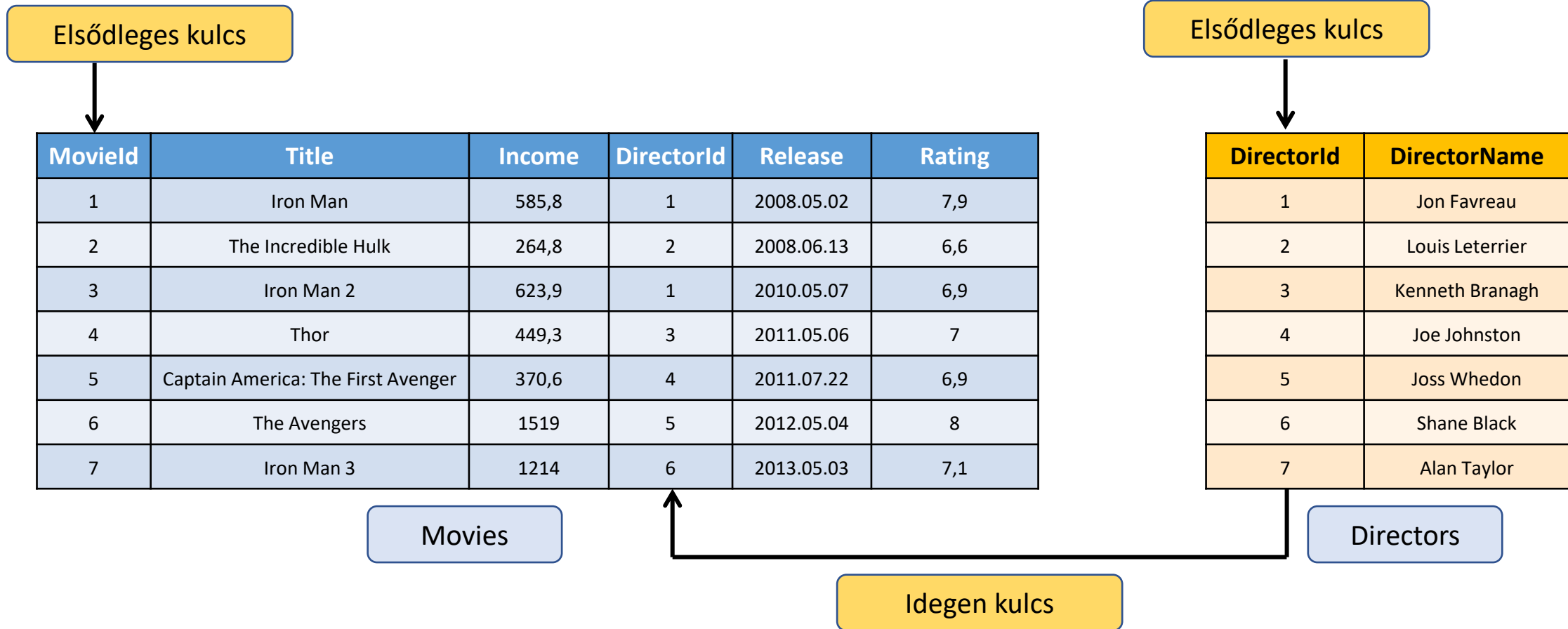
- Két megközelítés lehetséges az adatbázis kezelésére
- **Database-First**
 - **Van már adatbázisunk**
 - Fel van töltve adatokkal
 - Nincs feltöltve adatokkal, de a táblák ki vannak alakítva
 - Nincsenek még C# osztályaink
 - **Megoldás:** Adatbázis alapján generáltatunk C# osztályokat
 - **Tipikusan:** Van már adatbázis, amihez szükséges egy menedzser programot írni
- **Code-First**
 - **Nincs még adatbázisunk**
 - De vannak már C#-ban egyed osztályaink, amelyek gyűjteményeit adatbázisba szeretnénk menteni
 - **Megoldás:** C# osztályok alapján generáltatunk adatbázis táblákat
 - **Tipikusan:** Írjunk egy alkalmazást valamire, ami amúgy el is tudja menteni az adatokat

- **DbContext** leszármazott

- Adatbázist reprezentálja
- C# kóddal itt vannak leírva a megszorítások, elsődleges és idegen kulcsok
- **OnConfiguring()** metódus
 - Kapcsolat beállítása (connection string pl.)
- **OnModelCreating()** metódus
 - Táblák/entitások beállítása Fluent API segítségével
 - Kezdeti adatok elhelyezése a táblában (DbSeed)

- **DbSet<T> : IQueryable<T>**

- Egy táblát reprezentáló osztály
- Mintha egy egyszerű lista lenne egy tábla



- **MDF fájl létrehozása**

- Project → Add new item → Data → Service-based database → xyz.mdf
- Kijelölés → Property Editor ablak → Build Action : Content & Copy To... : Copy always

- **Adatbázis feltöltése és connection string megszerzése**

- View → Server Explorer → xyz.mdf jobb klikk → new sql query → SQL létrehozó kód beszúrása
- View → Server Explorer → xyz.mdf jobb klikk → Property Editor ablak → Connection String
 - Érdemes módosítani picit: AttachDbFilename=|DataDirectory|\movies.mdf
 - |DataDirectory| mindig a \bin\Debug\net5.0 mappát jelenti
- **Close Connection!**
- (Úgy vesszük, hogy egy valós projektben ez a lépés **már 1-2 éve elkészült** és csak egy connection stringet kapunk, ahol az adatbázis elérhető)

- **4 db szükséges nuget csomag telepítése (Core, Proxies, SqlServer, Tools)**

- **Generálatás**

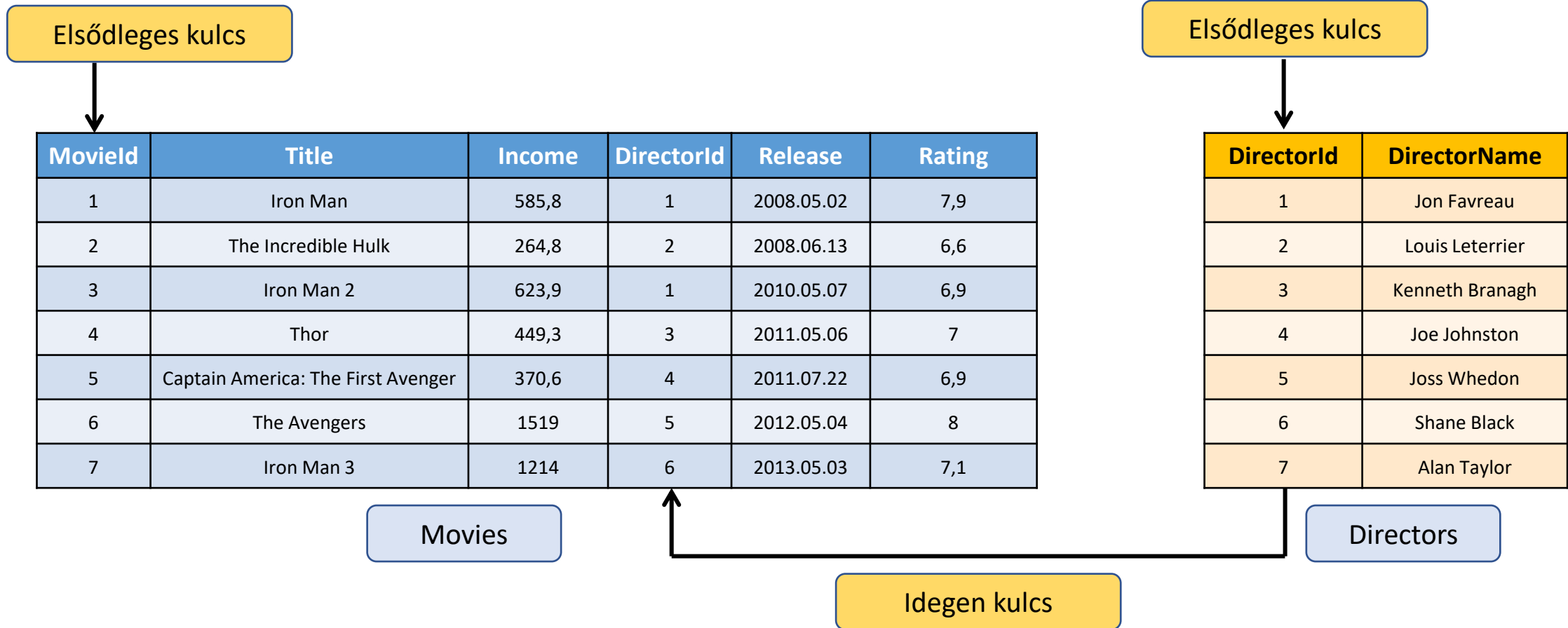
- Tools → Nuget Package Manager → Package Manager Console
- Utasítás beillesztése
 - Scaffold-DbContext "CONNECTION_STRING" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
- Ehhez a lépéshez kellett telepítenünk a **Microsoft.EntityFrameworkCore.Tools** csomagot

- Adatbázis példányosítása

```
MyDbContext ctx = new MyDbContext();
```

- Egy egyszerű lekérdezés

```
var movies = ctx.Movies.Where(t => t.Title.Contains("Avengers"));
```



- **MDF fájl létrehozása**

- Project → Add new item → Data → Service-based database → xyz.mdf
- Kijelölés → Property Editor ablak → Build Action : Content & Copy To... : Copy always

- **Connection string megszerzése**

- View → Server Explorer → xyz.mdf jobb klikk → Property Editor ablak → Connection String
 - Érdemes módosítani picit: AttachDbFilename=|DataDirectory|\movies.mdf
 - |DataDirectory| mindig a \bin\Debug\net5.0 mappát jelenti
- **Close Connection!**

- **4 db szükséges nuget csomag telepítése (Core, Proxies, SqlServer, Tools)**

- **Osztályok elkészítése és megszorításokkal ellátása**

- **DbContext leszármazott megírása**

- **Használat**

```
public class Movie
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int MovieId { get; set; }

    [StringLength(240)]
    public string Title { get; set; }

    [Range(0,10000)]
    public double Income { get; set; }

    [Range(0,10)]
    public double Rating { get; set; }

    public DateTime Release { get; set; }

    public int DirectorId { get; set; }
}
```

```
public class Director
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int DirectorId { get; set; }

    [Required]
    [StringLength(240)]
    public string DirectorName { get; set; }
}
```

- **[Key]**
 - Ez a tulajdonság legyen az elsődleges kulcs a táblában
- **[DatabaseGenerated(DatabaseGeneratedOption.Identity)]**
 - A tulajdonság értékét az adatbázis határozza meg → int kulcsnál megoldja a növelést eggyel!
- **[Required]**
 - Kötelező értéket adni neki, nem lehet null
- **[StringLength(240)]**
 - Ez lehet az adott string maximális hossza
 - Sokkal jobb lesz az adatbázis teljesítménye, mert akkor nvarchar(240) típusra alakít, különben text...
- **[Range(0,10)]**
 - Az adott szám csak ebben a tartományban lehet

- **[Column("id")]**
 - Az adatbázis mező neve legyen más, mint a tulajdonság neve
- **[Column(TypeName = "decimal(5, 2)")]**
 - Adatbázis típus explicit megadása
- **[ForeignKey(nameof(Director))]**
 - Az adott mező egy idegen kulcs adott típusra
- **[NotMapped]**
 - Az adott jellemző NE kerüljön adatbázisba, csak futásidőben van rá szükség
- **[JsonIgnore]**
 - JSON szerIALIZÁCIÓNÁL ne kerüljön bele a JSON-be


```
public class MovieDbContext : DbContext
{
```

```
    public DbSet<Movie> Movies { get; set; }
    public DbSet<Director> Directors { get; set; }
```

← táblák létrehozása

```
    public MovieDbContext()
    {
        this.Database.EnsureCreated();
    }
```

← létrehozó utasítás

```
    protected override void OnConfiguring(DbContextOptionsBuilder builder)
    {
        if (!builder.IsConfigured)
        {
            string conn = @"Data Source=(LocalDB)\MSSQLLocalDB;
                           AttachDbFilename=|DataDirectory|\movies.mdf;Integrated Security=True;MultipleActiveResultSets=true";


            builder
                .UseSqlServer(conn);
        }
    }
}
```

← SQL server elérés beállítása

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
```

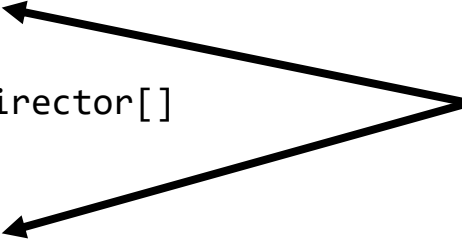
```
    modelBuilder.Entity<Movie>(movie => movie
        .HasOne<Director>()
        .WithMany()
        .HasForeignKey(movie => movie.DirectorId)
        .OnDelete(DeleteBehavior.Cascade));
```

Táblák közti
kapcsolatok beállítása



```
    modelBuilder.Entity<Movie>().HasData(new Movie[]
    {
        new Movie("1#Iron Man#585,8#1#2008*05*02#7,9"),
        new Movie("2#The Incredible Hulk#264,8#2#2008*06*13#6,6"),
        new Movie("3#Iron Man 2#623,9#1#2010*05*07#6,9")
    });
```

SEED adatok
(DB feltöltése kódból)



```
    modelBuilder.Entity<Director>().HasData(new Director[]
    {
        new Director("1#Jon Favreau"),
        new Director("2#Louis Leterrier"),
        new Director("3#Kenneth Branagh")
    });
```

```
}
```

- Az adatbázis létrehozásakor már beletöltünk „némi” adatot
- Az MDF fájl minden build esetén bemásolódik a **\bin\Debug\net5.0** mappába
 - Copy Always-el ezt állítottuk be
- Mindig felülírja az előzőleg bemásolt MDF fájlt
- Mindig belekerülnek újra a seed adatok
- Itt **mindig meg kell adnunk explicit az id-t**, ide nem vonatkozik a **[DatabaseGenerated]** attribútum beállítása
- Miért jó ez?
 - Fejlesztés közben SOSEM az éles adatbázissal dolgozunk
 - A DbSeed mindig egy kiinduló pont, mintha lenne éles adatbázis meglévő adatokkal
 - Ha véletlenül kitöröljük a tartalmát, akkor is a következő buildeléskor ott lesznek a seed adatok
 - Fejlesztés végén, ha minden tökéletesen működik, akkor átváltunk egy éles adatbázisra **pusztán azzal, hogy cseréljük a connection stringet**

- **OnModelCreating()**-ben állítottuk be

```
modelBuilder.Entity<Movie>(movie => movie
```

Egy **Movie** elem kapcsolatait állítjuk be

```
.HasOne<Director>()
```

Tartozik hozzá **EGY Director**

```
.WithMany()
```

Aki **TÖBB Movie**-hoz is tartozhat

```
.HasForeignKey(movie => movie.DirectorId)
```

A kapcsolódás a **Movie** elem **DirectorId** idegen kulcsa alapján történik

```
.onDelete>DeleteBehavior.Cascade));
```

Ha törölünk egy olyan **Directort**, akinek a kulcsa idegen kulcsként be van állítva egy **Movie** elemen, akkor **töröljük a Movie-t is!**

- **DeleteBehavior** lehetőségei

- **Restrict**: megakadályozás → Exception
- **SetNull**: idegen kulcs null-ra állítása (ha egyáltalán nullázható! Int? Típus)
- **Cascade**: töröljük a hivatkozó elemeket is

- Adatbázis példányosítása

```
MovieDbContext ctx = new MovieDbContext();
```

- Egy egyszerű lekérdezés

```
var movies = ctx.Movies.Where(t => t.Title.Contains("Avengers"));
```

- JOIN lekérdezést csinálunk az idegen kulcs alapján

```
var q = from x in ctx.Movies
        join y in ctx.Directors on x.DirectorId equals y.DirectorId
        select new
        {
            DirectorName = y.DirectorName,
            Title = x.Title
        };
```

- A JOIN lekérdezések elkerülhetőek **LazyLoading** használatával
 - Definiáljunk a **Movie** osztályba a **DirectorId**-n kívül egy **virtual Director** tulajdonságot
 - Definiáljunk a **Director** osztályba egy **virtual ICollection<Movie>** gyűjteményt!
 - Az Entity Framework képes futásidőben virtuálisan odahelyezni a kapcsolódó elemeket!
 - Némi beállítás után...
- Az előző lekérdezés LazyLoading-al

```
var q = from x in ctx.Movies
        select new
        {
            DirectorName = x.Director.DirectorName,
            Title = x.Title
        };
```

Nem kell a JOIN, hiszen a Movie-nak van egy virtuális Director tulajdonsága, amiben futásidőben megjelenik a kapcsolt Director entitás

- Ehhez van szükségünk a már telepített **Microsoft.EntityFrameworkCore.Proxies** csomagra
- **Movie** osztály kiegészítése

```
public virtual Director Director { get; set; }
```

- **Director** osztály kiegészítése

```
public virtual ICollection<Movie> Movies { get; set; }
```

```
public Director()  
{  
  
}  
}
```

```
    Movies = new HashSet<Movie>();
```

Egy valós gyűjteményként implementáljuk

- Connection string végére kell: **MultipleActiveResultSets = true**
- OnConfiguringbe: **UseLazyLoadingProxies()**

- A **DbContext** osztályban módosítuk az **OnModelCreating()** metódust

```
modelBuilder.Entity<Movie>(movie => movie
```

```
    .HasOne(movie => movie.Director)
```

```
    .WithMany(director => director.Movies)
```

```
    .HasForeignKey(movie => movie.DirectorId)
```

```
    .OnDelete(DeleteBehavior.Cascade));
```

Egy **Movie** elem kapcsolatait állítjuk be

Tartozik hozzá **EGY Director**, amely megjelenik a virtuális **Director** tulajdonságban futásidőben

És a **Director** osztályban a hozzá tartozó **Movie**-k megjelennek **Movies** néven gyűjteményként

- Ezeket a virtuálisan LazyLoadinggal odamappelt tulajdonságokat **Navigation Property**-nek nevezzük

- A **LazyLoading** helyett használhatunk **EagerLoading**-ot is

- Teljesítményben jobb
- Előre tudni kell, hogy mit akarunk mappelni és hova
- Körülményesebb használni

- **Kód**

```
foreach (var item in ctx.Directors.Include(t => t.Movies))
{
    Console.WriteLine(item.DirectorName);
    foreach (var movie in item.Movies)
    {
        Console.WriteLine("\t" + movie.Title);
    }
}
```

Include segítségével kell megadnunk, hogy a virtuális gyűjtemény ki legyen töltve elemekkel

- Ekkor nem kell a **...Proxies** nuget csomag, **UseLazyLoadingProxies()** hívás sem és a connection stringben sem kell a **MultipleActiveResultSet = true**

- **Beszúrás**

```
Movie m = new Movie()  
{  
    Title = "Doctor Strange in the M.O.M",  
    DirectorId = 1  
};  
ctx.Movies.Add(m);  
ctx.SaveChanges();
```

- **Módosítás**

```
Movie m = ctx.Movies.FirstOrDefault(t => t.Title.Contains("Strange"));  
m.Title = "Doctor Strange in the Multiverse of Madness";  
ctx.SaveChanges();
```

- **Törlés**

```
Movie m = ctx.Movies.FirstOrDefault(t => t.Title.Contains("Strange"));  
ctx.Movies.Remove(m);  
ctx.SaveChanges();
```

Több-a-többhöz kapcsolat

44

MovielId	Title	Income	DirectorId	Release	Rating
1	Iron Man	585,8	1	2008.05.02	7,9
2	The Incredible Hulk	264,8	2	2008.06.13	6,6
3	Iron Man 2	623,9	1	2010.05.07	6,9
4	Thor	449,3	3	2011.05.06	7

Movies

ActorId	ActorName
46	Chris Hemsworth
73	Edward Norton
96	Gwyneth Paltrow
212	Robert Downey Jr.

Actors

Idegen kulcs

Idegen kulcs

RoleId	MovielId	ActorId	Priority	RoleName
1	1	212	1	Tony Stark
2	1	96	2	Pepper Potts
19	2	73	1	Bruce Banner
41	4	46	1	Thor

Roles kapcsolótábla

```
public class Role
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int RoleId { get; set; }

    public int Priority { get; set; }
    public string RoleName { get; set; }

    public int MovieId { get; set; }
    public int ActorId { get; set; }

    public virtual Actor Actor { get; private set; }
    public virtual Movie Movie { get; private set; }
}
```

```
public class Actor
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int ActorId { get; set; }

    [Required]
    [StringLength(240)]
    public string ActorName { get; set; }

    public virtual ICollection<Movie> Movies { get; set; }
}
```

```
public class Movie
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int MovieId { get; set; }

    public string Title { get; set; }

    public virtual ICollection<Actor> Actors { get; set; }
}
```

+többi tulajdonság

- **OnModelCreating()-ben:**

```
modelBuilder.Entity<Actor>() ← Egy Actor elem kapcsolatait állítjuk be
    .HasMany(x => x.Movies) ← Az Actor-nak van SOK Movie-ja
    .WithMany(x => x.actors) ← A Movie-nek van SOK Actor-ja
    .UsingEntity<Role>( ← A MovieId-ket és ActorId-ket a Role entitás kapcsolja össze
        x => x.HasOne(x => x.Movie) ← Aminek van EGY Movie-ja a MovieId alapján
            .WithMany().HasForeignKey(x => x.MovieId).OnDelete(DeleteBehavior.Cascade),
        x => x.HasOne(x => x.Actor) ← És van EGY Actor-ja az ActorId alapján
            .WithMany().HasForeignKey(x => x.ActorId).OnDelete(DeleteBehavior.Cascade));
```

- **DbContextben új táblák**

```
public DbSet<Role> Roles { get; set; }
public DbSet<Actor> Actors { get; set; }
```

- Érdemes lenne még
 - Virtuális gyűjteményként látni egy **Actor**-nál a **Role**-jait
 - Virtuális gyűjteményként látni egy **Movie**-nál a benne lévő **Role**-okat

```
modelBuilder.Entity<Role>()  
    .HasOne(r => r.Actor)  
    .WithMany(actor => actor.Roles)  
    .HasForeignKey(r => r.ActorId)  
    .OnDelete(DeleteBehavior.Cascade);
```

```
modelBuilder.Entity<Role>()  
    .HasOne(r => r.Movie)  
    .WithMany(movie => movie.Roles)  
    .HasForeignKey(r => r.MovieId)  
    .OnDelete(DeleteBehavior.Cascade);
```

- **Van interneten át publikus IP címmel elérhető SQL server**
 - Connection stringgel rákapcsolódunk
- **Létrehozunk a saját gépünkön/hálózatunkban SQL servert**
 - Connection stringgel rákapcsolódunk
- **LocalDb-vel MDF fájlokba dolgozunk**
 - Létre kell hozni az MDF és LDF fájlokat, a projekt részeként mozgatni kell
 - LocalDb csak Windowson van
 - Bele tudunk nézni a táblákba
- **InMemoryDatabase-t használunk**
 - Valójában az Entity Framework a memóriába hozza létre a táblákat
 - Semmilyen fájlt nem kell a projekttel mozgatni
 - Valójában nincs is fizikai adattárolás → de bármikor áttérhetünk 1 sor módosításával
 - Nem tudunk a táblákba belenézni

- Mikor lesz rá szükségünk?
 - Féléves feladatnál
 - MAC-en dolgozva
- Mikor lehet használni?
 - Code-First megközelítés esetén
- Használat
 - Nuget csomag telepítése
 - Microsoft.EntityFrameworkCore.InMemory
 - OnConfiguring()-ben bekapcsolás

```
builder
    .UseInMemoryDatabase("mydb")
    .UseLazyLoadingProxies();
```

- Hajtsuk végre az alábbi lekérdezéseket!
 - Hány filmben szerepel Robert Downey Jr.?
 - Melyik a legtöbb szereplőt felsorakoztató film?
 - Írjuk ki Paul Rudd szerepeit és filmjeit!
 - Évente milyen átlagértékelésű filmek születtek?



Köszönöm a figyelmet!

Kérdés esetén e-mailben szívesen állok rendelkezésre.