# Töltsünk fel adatokkal egy N:M-es adatbázist, majd írjunk lekérdezéseket

Ebben a részben az eddigi ismereteinket kiegészítve feltöltünk fájlokból egy **N:M**-es adatbázist (*Szinesz* <=> *Film*)


Először a táblákat reprezentáló osztályokat kell létrehozni, figyelve arra, hogy **1 színész több filmben is játszhat és 1 filmben több színész is játszhat**. Csak úgy tudjuk megoldani, hogy létrehozunk egy **kapcsolótáblát**, ahol szerepel majd a **SzineszId** és a **FilmId** is és ez a **kettő együtt lesz a kulcs** (*kompozit kulcs!*)!

Pl. az 1-es **Id**-vel rendelkező színész filmjeit fel tudjuk majd sorolni így: (1,5), (1,8), (1,7), ahol az 5, 8, 17 azoknak a filmeknek az **Id**-t jelenti, amelyekben játszott a színész és nincs az adatbázisban több olyan film, amelyben játszott volna!

```
public class Szinesz
{
    0 references
    public int SzineszId { get; set; }
    0 references
    public string Nev { get; set; } = string.Empty;
    0 references
    public int SzulEv { get; set; }
    0 references
    public string Nemzetiseg { get; set; } = string.Empty;
    0 references
    public string Nem { get; set; } = string.Empty;
    // Navigációs property a kapcsolótáblához - M:N
    1 reference
    public ICollection<FilmSzinesz> Filmek { get; set; } = new List<FilmSzinesz>();
}
4 references
public class Film
{
    0 references
    public int FilmId { get; set; }
    0 references
    public string Cim { get; set; } = string.Empty;
    0 references
    public int KiadasEve { get; set; }
    0 references
    public string Mufaj { get; set; } = string.Empty;
    0 references
    public string Rendezo { get; set; } = string.Empty;
    // Navigációs property a kapcsolótáblához - M:N
    1 reference
    public ICollection<FilmSzinesz> FilmSzineszek { get; set; } = new List<FilmSzinesz>();
}
// Létrehozzuk a kapcsolótáblát reprezentáló osztályt
8 references
public class FilmSzinesz
{
    2 references
    public int SzineszId { get; set; }
    1 reference
    public Szinesz Szinesz { get; set; } = null!;

    2 references
    public int FilmId { get; set; }
    1 reference
    public Film Film { get; set; } = null!;
}
```

Mint látható, a navigációs property-k a **Szinesz** és **Film** osztályokban listák lesznek, hisz **egy színesz => sok film; egy film => sok színész**.

A **FilmSzinesz (EBBŐL LESZ A KAPCSOLÓTÁBLA)** osztályban viszont egy **SzineszId**-hoz betöltjük a **Szinesz** osztályból származó **Szinesz** nevű objektumot, ami tartalmazza az adott **Id**-vel rendelkező színész minden adatát! És ugyan ezt tesszük a **Film**-el is! A **Fluent API**-nál ezt fogjuk felhasználni az összekapcsoláshoz!

Hozzuk is létre a kapcsolatot az **AppDbContext** osztályban az **OnModelCreating** metódus felülírásával.

Először létrehozzuk a kompozit kulcsot (összetett kulcs, itt a két mező értékének **EGYÜTT KELL EGYEDINEK LENNIE**!)

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<FilmSzinesz>()
        .HasKey(fs => new {fs.SzineszId, fs.FilmId});
}
```

Majd nézzük meg az összekapcsolást, amit utólag egyszerűbb magyarázni, mint előre. **FONTOS**, hogy itt két kapcsolatot kell definiálni!

```
modelBuilder.Entity<FilmSzinesz>()
    .HasOne(fs => fs.Film)
    .WithMany(f => f.FilmSzineszek)
    .HasForeignKey(fs => fs.FilmId);

modelBuilder.Entity<FilmSzinesz>()
    .HasOne(fs => fs.Szinesz)
    .WithMany(s => s.Filmek)
    .HasForeignKey(fs => fs.SzineszId);
```

A **FilmSzinesz** a kapcsolótábla, erre definiálom az **1:N**-es kapcsolatokat mindkét oldalról.

A kapcsolóosztályban lévő **Film** objektumhoz, ami egy film adatait tartalmazza, hozzákapcsolom az **ÖSSZES SZÍNÉSZT**, aki játszik abban a filmben! Az idegen kulcs a kapcsolótáblában a **FilmId:**

```
modelBuilder.Entity<FilmSzinesz>()
    .HasOne(fs => fs.Film)
    .WithMany(f => f.FilmSzineszek)
    .HasForeignKey(fs => fs.FilmId);
```

De így még csak a filmekben játszó színészeket találom meg, a színészekhez tartozó filmeket még nem tudjuk leszűrni, ehhez kell a második kapcsolat!

Ebben a második kapcsolatban a kapcsolóosztályban lévő **Szinesz** objektumhoz, ami egy színész adatait tartalmazza, hozzákapcsolom az **ÖSSZES FILMET**, amelyben az adott színész játszott. Az idegen kulcs a kapcsolótáblában a **SzineszId**.

Ekkor már le tudom szűrni a színészekhez tartozó filmeket is!

Nézzük egyben az egészet:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<FilmSzinesz>()
        .HasKey(fs => new {fs.SzineszId, fs.FilmId});

    modelBuilder.Entity<FilmSzinesz>()
        .HasOne(fs => fs.Film)
        .WithMany(f => f.FilmSzineszek)
        .HasForeignKey(fs => fs.FilmId);

    modelBuilder.Entity<FilmSzinesz>()
        .HasOne(fs => fs.Szinesz)
        .WithMany(s => s.Filmek)
        .HasForeignKey(fs => fs.SzineszId);
}
```

Innentől már csak migrálni kell, majd beolvasni az adatokat a fájlból egy listába és fel kell vele tölteni az adatbázist. Nosza! Megint figyelni kell a sorrendre:
- **Filmek** és **Szineszek** és csak utána a kapcsolótábla, hogy legyen értéke a **SzineszId** és **FilmId**-nak!

Mivel kiderült, hogy a feltöltés nem gyerekjáték, kiteszem ide is az egészet. Bár itt nem bontottam ki minden lépését, hanem igyekeztem röviden írni, de azért érthetőnek kellene lennie!

Létrehoztam három listát, ezek tartalmazzál majd a fájl adatait, ezekkel töltöm majd fel a táblákat:

```
List<Szinesz> s = new();
List<Film> f = new();
List<FilmSzinesz> fs = new();
```

Utána végigmegyek minden fájlon és az adataikat beolvasom a listába. A **navigációs propertyvel** itt nem kell foglalkoznom, mert az osztály konstruktorában már létrehoztam, a feltöltését majd automatikusan elintézi a **EF**! Ez lesz a **Szinesz** tábla:

```
foreach (var i in File.ReadAllLines("szinesz.csv").Skip(1))
{
    var resz = i.Split(';');
    s.Add(new Szinesz
    {
        SzineszId = int.Parse(resz[0]),
        Nev = resz[1],
        SzulEv = int.Parse(resz[2]),
        Nemzetiseg = resz[3],
        Nem = resz[4]
    });
}
```

Ez a **Film** tábla:

```
foreach (var i in File.ReadAllLines("film.csv").Skip(1))
{
    var resz = i.Split(';');
    f.Add(new Film
    {
        FilmId = int.Parse(resz[0]),
        Cim = resz[1],
        KiadasEve = int.Parse(resz[2]),
        Mufaj = resz[3],
        Rendezo = resz[4]
    });
}
```

Ez pedig a kapcsoló tábla:

```
foreach (var i in
File.ReadAllLines("mozi_kapcsolat.csv").Skip(1))
{
    var resz = i.Split(';');
    fs.Add(new FilmSzinesz
    {
        SzineszId = int.Parse(resz[0]),
        FilmId = int.Parse(resz[1])
    });
}
```

Utána már csak arra kell figyelni, hogy az adatok feltöltése jó sorrendben történjen, tehát először a függőségek nélküli táblákat töltjük fel és csak utána jöhet a kapcsolatot megvalósító tábla, hogy ki tudja szedni a **SzineszId** és **FilmId** értékeket! (nem szükséges külön menteni, elég csak a végén, de így most látványosabb és érthetőbb!)

Valahogy így:

```
using var ctx = new AppDbContext();

ctx.Szineszek.AddRange(s);
ctx.Filmek.AddRange(f);
ctx.SaveChanges();
ctx.FilmekSzineszek.AddRange(fs);
ctx.SaveChanges();
```

## Lekérdezések

Ha kész az adatbázis, akkor lekérdezhetünk belőle pár dolgot. A **Program**.cs-ben kikommentelhetjük a sorokat, egyedül az **AppDbContext** osztály példányosítását hagyjuk meg!

```
var ctx = new AppDbContext();
```

Kezdjünk egyszerűvel, amihez elég a **Filmek** tábla használata. Az összes film:

```
var filmek = ctx.Filmek;
foreach (var f in filmek)
{
    Console.WriteLine(f.Cim);
}
```

Az összes színész, amihez elég a **Szineszek** tábla használata:

```
var szineszek = ctx.Szineszek;
foreach (var f in szineszek)
{
    Console.WriteLine(f.Nev);
}
```

Utána nézzünk kicsit bonyolultabbat. **EZT MEGINT MEG KELLENE ÉRTENED!** Többféleképpen kapcsoljuk össze a táblákat és mindig arra a kérdésre válaszolunk, hogy melyik színész melyik filmben játszott.

Először a kapcsolótáblából indulunk (***FilmekSzineszek***) Itt ugyebár a navigációs propertyk a **Szinesz** és a **Film** osztályból származó objektumok, könnyű **Include**.

```
var adatok = ctx.FilmekSzineszek
        .Include(fs => fs.Szinesz)
        .Include(fs => fs.Film);
```

Az összekapcsolás után végigszaladunk a gyűjteményen és kigyűjtjük a minket érdeklő adatokat:

```
var adatok = ctx.FilmekSzineszek
        .Include(fs => fs.Szinesz)
        .Include(fs => fs.Film);

// melyik színész melyik filmben játszik?
foreach (var i in adatok)
{
    Console.WriteLine($"{i.Szinesz.Nev} - {i.Film.Cim}");
}
```

Most a Filmek osztályból indulunk. Itt a navigációs property egy lista a kapcsolótábla objektumaiból (***FilmSzineszek***), majd a listában lévő objektumból (egy szinttel beljebb, ezért a **ThenInclude**) tudjuk kinyerni a **Szinesz** adatait:

```
var lista = ctx.Filmek
    .Include(f => f.FilmSzineszek)
        .ThenInclude(fs => fs.Szinesz);

foreach (var film in lista)
{
    foreach (var sz in film.FilmSzineszek)
        Console.WriteLine($"{sz.Szinesz.Nev} - {film.Cim}");
}
```

Most a **Szineszek** osztályból indulunk. Itt a navigációs property egy lista a kapcsolótábla objektumaiból (***Filmek***), majd a listában lévő objektumból (egy szinttel beljebb, ezért a **ThenInclude**) tudjuk kinyerni a **Film** adatait:

```
var lista1 = ctx.Szineszek
    .Include(s => s.Filmek)
        .ThenInclude(fs => fs.Film);

foreach (var sz in lista1)
{
    foreach (var s in sz.Filmek)
        Console.WriteLine($"{sz.Nev} - {s.Film.Cim}");
}
```

Ezekkel az **Include**-okkal a teljes objektumgráfot betöltjük. Bármilyen adatot megszerezhetünk a példában szereplő **LISTA** és **LISTA1** gyűjteményekből, hisz bennük van minden tábla minden adata. Épp ez a hátránya több tábla esetén az **Include**-nak! Túl összetett, túl sok adat.
Ha csak adatot akarunk kinyerni akkor használjuk a **SELECT** utasítást, ami **SQL** szinten **JOIN**-t generál és **NEM KELL HOZZÁ INCLUDE**! Cserébe *gyors*, viszont **DTO** szerűen csak a megfogalmazott kérésnek megfelelő adatokat tölti le. Ahogy az előbb írtam, **SQL**-ben generál **JOIN**-t. Nézzük az előző példát ezzel a módszerrel a kapcsolótábla felől:

```
var film_szer = ctx.FilmekSzineszek
    .Select(s => new
    {
        nev = s.Szinesz.Nev,
        film = s.Film.Cim
    });
foreach (var z in film_szer)
{
    Console.WriteLine($"{z.nev} - {z.film}");
}
```

Most nézzük a **Szinesz** tábla felől:

```
var film_szer1 = ctx.Szineszek
    .Select(x => new
    {
        nev = x.Nev,
        film = x.Filmek.Select(y => y.Film.Cim)
    });

foreach (var z in film_szer1)
{
    foreach (var i in z.film)
    {
        Console.WriteLine($"{z.nev} - {i}");
    }
}
```

Itt arra kell figyelni, hogy az első **SELECT**-ben a ***film*** érték egy **LISTA**, hisz egy színész sok filmben játszhat, tehát a kiírásnál végig kell rajta szaladni egy ciklussal!

Végezetül nézzük a **Filmek** tábla felől:

```
var film_szer2 = ctx.Filmek
    .Select(x => new
    {
        nev = x.FilmSzineszek.Select(y => y.Szinesz.Nev),
        film = x.Cim
    });

foreach (var z in film_szer2)
{
    foreach (var i in z.nev)
    {
        Console.WriteLine($"{i} - {z.film}");
    }
}
```

Itt arra kell figyelni, hogy az első **SELECT**-ben a ***nev*** érték egy **LISTA**, hisz egy filmben sok színész játszhat, tehát a kiírásnál végig kell rajta szaladni egy ciklussal!

Viszont így macerás a kiírás, mert vagy a filmek vagy a színészek adatai listában jönnek vissza. Ezen segít a ***SelectMany()***
pl.:

```
var sokadik = ctx.Filmek
    .SelectMany(x => x.FilmSzineszek)
    .Select(x => new
    {
        nev = x.Szinesz.Nev,
        cim = x.Film.Cim
    }
    );
```

és ilyenkor a kiírás leegyszerűsödik:

```
foreach (var z in sokadik)
{
    Console.WriteLine($"{z.nev} - {z.cim}");
}
```

Itt ez a sor:

```
.SelectMany(x => x.FilmSzineszek)
```

azt csinálja, hogy minden filmhez tartozó **FilmSzinesz** rekordot **külön sorra bont,** ha egy filmben 5 színész van, akkor 5 rekordot ad vissza. Nem kell majd ciklusokat egymásba ágyazni a kiíráshoz!

Így itt minden sor egy **Film** – **Szinesz** páros

```
.Select(x => new
    {
        nev = x.Szinesz.Nev,
        cim = x.Film.Cim
    }
```

Összefoglalva a lényeget. Az **EF Core** a **Select**-ből tudja, hogy **csak a** *nev* **és** *cim* **kell**, így nem tölti le feleslegesen a teljes entitásokat. Minél nagyobb egy adatbázis, annál jobban kijön, hogy ez a módszer sokkal gyorsabb, így ha célzottan valamilyen adatra vagyunk kíváncsiak és nem kell a teljes objektumgráf (minden adat), akkor a **SELECT** lesz a jó választás **SelectMany** használattal kiegészítve!

Talán még megvan, hogy az előző anyagban feltettem egy kis mintát táblák összekapcsolására **INCLUDE NÉLKÜL SELECT**-el. Ennek a szintaxisa erősen hasonlít az **SQL** szintxisára és a neve:
**Query syntax (from … join … select …)**
kinézete**:**

```
var adatok =
    from x in ctx.Versenyzok
    join s in ctx.Szakmak on x.SzakmaId equals s.Id
    join y in ctx.Orszagok on x.OrszagId equals y.Id
    select new
    {
        Id = x.Id,
        Nev = x.Nev,
        Pont = x.Pont,
        Szakma = s.SzakmaNev,
        Orszag = y.OrszagNev,
    };

foreach (var i in adatok.OrderByDescending(x=>x.Pont).Take(10))
{
    Console.WriteLine($"{i.Nev} - {i.Szakma} - {i.Orszag} - {i.Pont}");
}
```

Az előbb mutatott **LINQ**-s **SELECT**-ek szintaxisa pedig a **Method syntax (Select / LINQ lambdákkal).** Bizonyos mennyiségű **LINQ** után ezt a legegyszerűbb használni, de ízlés dolga. pl az előbb:

```
var film_szer2 = ctx.Filmek
    .Select(x => new
    {
        nev = x.FilmSzineszek.Select(y => y.Szinesz.Nev),
        film = x.Cim
    });
```

A kettő között az a fő különbség, hogy **Method szintaxis** esetén az **SQL JOIN**-t automatikusan generálja az **EF**, míg **Query szintaxis** esetén a programozó felügyeli!

Nagyjából ennyit szerettem volna mondani erről az anyagrészről, most még nézzünk pár mintafeladatot:

Egy film összes színésze (**Cim** alapján):

```
var szineszek = ctx.Filmek
    .Where(x => x.Cim == "Titanic")
    .SelectMany(x => x.FilmSzineszek)
    .Select(y => y.Szinesz.Nev);
// kiírás
foreach (var z in szineszek)
{
    Console.WriteLine(z);
}
```

Hány színész játszott egy filmben?

```
var db = ctx.Filmek
    .Select(x => new
    {
        film = x.Cim,
        db = x.FilmSzineszek.Count
    }
    );
//kiírás
foreach(var z in db)
{
    Console.WriteLine($"{z.film} - {z.db}");
}
```

Egy színész hány filmben szerepelt?

```
var szdb = ctx.Szineszek
    .Select(x => new
    {
        nev = x.Nev,
        db = x.Filmek.Count
    }
    );
//kiírás
foreach (var z in szdb)
{
    Console.WriteLine($"{z.nev} - {z.db}");
}
```

Ki játszott a legtöbb filmben?

```
var legtobb = ctx.Szineszek
    .Select(x => new
    {
        nev = x.Nev,
        db = x.Filmek.Count
    })
    .OrderByDescending(x => x.db)
    .FirstOrDefault();
// kiírás
Console.WriteLine($"{legtobb!.nev} - {legtobb.db}");
```

És a végére nézzünk egy kellően „dögös” lekérdezést. Kik játszottak együtt Morgan Freeman-el?

```
var egyutt = ctx.FilmekSzineszek
    .Where(x => x.Szinesz.Nev == "Morgan Freeman")
    .SelectMany(x => x.Film.FilmSzineszek)
    .Where(x => x.Szinesz.Nev != "Morgan Freeman")
    .Select(x => x.Szinesz.Nev)
    .Distinct();
// kiírás 2 különböző módon
Console.WriteLine(string.Join("; ", egyutt));
foreach (var z in egyutt)
{
    Console.WriteLine(z);
}
```

Hát ez ennyi volt. Jó tanulást mindenkinek!