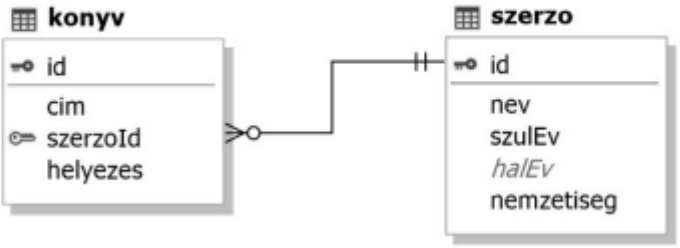
Kéttáblás adatbázis létrehozása Code First verzióban migrálással

Az adattáblák közti kapcsolatokat az alábbi ábra mutatja:



Elkészítjük a **Konyv** osztályt és ebbe a fájlba hozzuk létre a **Szerzo** osztályt is.

A táblák között 1 : N kapcsolat van!!!

Egy könyvet csak 1 szerző írhatott, de 1 szerző N könyvet írhatott!

A **Konyv** osztály tulajdonságainál használtam attribútumokat.
Az egyedüli újdonság az **Idegen Kulcs (FK)** Az első példánál a ritkább mód látható és általában az EF automatikusan felismeri az **Id** végű propertykben az idegen kulcsot, tehát ha nem teszünk rá attribútumot, akkor is működni fog:

```
0 references
public class Konyv
{
    0 references
    public int Id { get; set; }

    [Required]  // not null
    [StringLength(100)]  // max 100 karakter hosszú lehet
    0 references
    public string Cim { get; set; }

    // Külső kulcs az egyik módon használva
    [ForeignKey("Szerzo")]
    0 references
    public int SzerzoId { get; set; }

    [Required]
    [Range(1, 200)]
    0 references
    public int Helyezes { get; set; }

    // Navigációs tulajdonság, a segítségével a Könyv osztályban is megismerhetjük a Szerzőt!
    0 references
    public Szerzo Szerzo { get; set; }
}
```

A másik mód, amit gyakrabban használunk:

```
public class Konyv
{
    2 references
    public int Id { get; set; }

    [Required]  // not null
    [StringLength(100)]  // max 100 karakter hosszú lehet
    3 references
    public string Cim { get; set; } = string.Empty;

    4 references
    public int SzerzoId { get; set; }

    [Required]
    [Range(1, 200)]
    2 references
    public int Helyezes { get; set; }

    // Navigációs tulajdonság, a segítségével a Könyv osztályban is megismerhetjük a Szerzőt!
    // + Külső kulcs a másik módon
    [ForeignKey(nameof(SzerzoId))]
    2 references
    public Szerzo Szerzo { get; set; } = null!;
```

A navigációs tulajdonság azt jelenti, hogy létrehozunk a **Konyv** osztályban egy objektumot a **Szerzo** osztályból, így a **Kony** osztályban is mindent megismerhetünk a **Szerzőről** ennek az objektumnak a segítségével***! Elég egyetlen osztály, hogy mindkét osztály információit használhassuk.***

```
public class Szerzo
{
    2 references
    public int Id { get; set; }
    4 references
    public string Nev { get; set; } = string.Empty;
    2 references
    public int Szulev { get; set; }
    2 references
    public int? Halev { get; set; }
    4 references
    public string Nemzetiseg { get; set; } = string.Empty!;
    // Navigációs tulajdonság. Egy gyűjtemény kell, hisz egy Szerző több Könyvet is írhatott!
    5 references
    public List<Konyv> Konyvek { get; set; } = null!;
```

A **Szerzo** osztályban, ahol **N** db könyv tartozhat **1** szerzőhöz, már egy gyűjtemény kell a **Navigációs Property** megvalósításához! Ez általában **Lista**! A gyűjteménybe belekerül az adott szerző összes könyve objektumként, így ebben az osztályban is benne lesz minden információ!

A kapcsolati osztályt (**AppDbContext** általában a neve) úgy hozzuk létre, hogy 2 táblát definiálunk benne és most a **Migrálás** miatt **nem kell benne külön konstruktort is létrehozni**! A migrálás miatt az adatbázis szerkezete a későbbiekben is módosítható lesz!!!

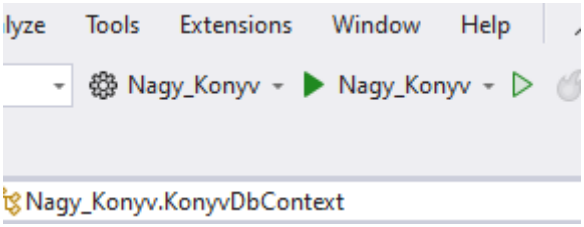
```
namespace Nagy_Konyv
{
    0 references
    public class KonyvDbContext:DbContext
    {
        0 references
        public DbSet<Konyv> Konyvek { get; set; }
        0 references
        public DbSet<Szerzo> Szerzok { get; set; }

        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            base.OnConfiguring(optionsBuilder);
            optionsBuilder.UseMySQL("server=localhost;database=nagykonyv;uid=root;password='';");
        }
    }
}
```

Nagyon FONTOS, hogy a migráláshoz kell a **Microsoft.EntityFrameworkCore.Tools** névtér!

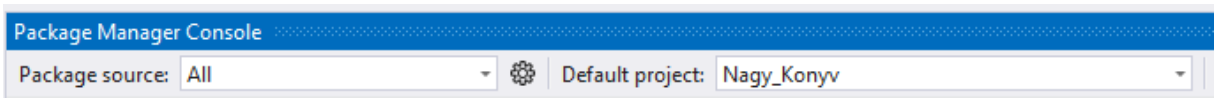
Migrálni több módon is lehet, de a legegyszerűbb a **Package Manager Console**-ból indítva!

Tools → NuGet Package Manager → Package Manager Console útvonalon érhető el az ablak.

Nagyon **FONTOS**, hogy itt:



és itt:



a **PROJEKTÜNK NEVE** legyen beállítva (ebben van ugye a **DbContext**-et tartalmazó osztály). Jelen esetben a **Nagy_Konyv**!!!

Migráció létrehozása a PM konzolban:

Add-Migration InitialCreate

Itt a parancs az **Add-Migration** és a**z InitialCreate** az első migráció egyezményes neve. A parancs létrehoz egy új mappát – *2025xxxxxx_InitialCreate.cs* néven, amely az adatbázis létrehozásának lépéseit tartalmazza.

A migráció létrehozása után létre kell hozni magát az adatbázist az
Update-Database
paranccsal és készen is vagyünk. Már csak adat kell bele!

Adatok feltöltése a **Szerzo** osztály irányából. Ne feledjük, hogy a **Listát** is fel kell töltenünk, ha több könyvet is írt!!!

Mindenképp **MEG KELL ÉRTENED**, hogy mi miért történik itt!

```
using KonyvDbContext ctx = new KonyvDbContext();

var iro = new Szerzo
{
    Nev = "Agatha Christie",
    Szulev = 1890,
    Halev = 1976,
    Nemzetiseg = "angol",
    Konyvek = new List<Konyv>
    {
        new Konyv {
            Cim = "Tíz kicsi néger",
            SzerzoId = 1,
            Helyezes = 3
        },
        new Konyv {
            Cim = "Húsz kicsi néger",
            SzerzoId = 1,
            Helyezes = 5
        }
    }
};
ctx.Szerzok.Add(iro);
ctx.SaveChanges();
```

Adatok feltöltése a **Konyv** osztály irányából (általában az **1:N**-es adatbázisok esetében az **N** irányából egyszerűbb megérteni a navigációs property-k feltöltését!):

```
var mu = new Konyv
{
    Cim = "Vuk",
    SzerzoId = 2,
    Helyezes = 2,
    Szerzo = new Szerzo
    {
        Nev = "Fekete István",
        Szulev = 1900,
        Halev = 1970,
        Nemzetiseg = "magyar"
    }
};
ctx.Konyvek.Add(mu);
var mu1 = new Konyv
{
    Cim = "Egypercesek",
    SzerzoId = 3,
    Helyezes = 11,
    Szerzo = new Szerzo
    {
        Nev = "Róka Sándor",
        Szulev = 1961,
        Halev = null,
        Nemzetiseg = "magyar"
    }
};
ctx.Konyvek.Add(mu1);
ctx.SaveChanges();
```

Járjuk kicsit körbe a MIGRÁLÁST!

Nagyon ajánlott, hogy az összes osztályban, amelyikből tábla lesz, hozzunk létre egy **ÜRES KONSTRUKTORT**! Ugyanis ha migrálni szeretnénk, akkor **Navigációs property-t NEM** rakhatunk a konstruktorba, mert az EF Navigációs property-t **nem** tud konstruktorból beállítani! Viszont ha van üres konstruktor, akkor azt tudja használni az EF, te meg használod a Navigációs property-t tartalmazó konstruktort, amire akarod!

Ha **ÚJ PROPERTY**-t (pl. a **Konyv** osztályba egy **Oldalszam** property-t) vettem fel az osztályba, amelyből tábla lesz, akkor **UPDATE**-et kell végrehajtanom, hogy ez a változás megjelenjen az adatbázisban is!
FONTOS, hogy Nem lehet "update-elni" a régi migrációt
A meglévő migrációkat elvileg lehet kézzel módosítani, de **NEM** ajánlott (és hibákhoz vezet).
EF Core elve: **minden modellváltozáshoz új migrációt kell létrehozni!**

Az **UPDATE** végrehajtása**:**
Add-Migration „Az új migráció neve, ami utal az új property-re”

jelen esetben pl.: **Add-Migration *AddNewPropertyToKonyv***

Majd FRISSÍTJÜK az adatbázist a következő paranccsal:

Update-Database

Utolsó migráció törlése (visszavonása):
Ha még nem alkalmaztuk az adatbázisra (nem adtuk ki az **Update-Database** parancsot) akkor így törölhetjük a migrációs fájlokat:

Remove-Migration

Ha már alkalmaztuk az adatbázisra, akkor előbb vissza kell állítani az előző állapotra, az ezt megelőző **Migrációra**, majd utána kell törölni:

Update-Database *previousMigrationName*
Remove-Migration

Visszalépés egy korábbi migrációra:
Ez visszaviszi az adatbázist egy adott korábbi migráció állapotára:

Update-Database *MigrationName*

Például**: Update-Database *InitialCreate***

Adatbázis tábláinak törlése (maga az adatbázis megmarad, de tábla nélkül):

Update-Database 0

Végezetül nézzük meg az adatbázis feltöltését fájlokból.

Most két fájl lesz a két táblához. Először beolvassuk a fájlokat egy-egy listába. **DE** figyelni kell arra, hogy az **1:N** kapcsolat miatt az objektumok felépítése egy picit rafináltabb.

A **Navigációs Property létrehozása a macerás kicsit!**

Az **N** ágon lévő osztály objektumában csak létrehozzuk, de **NEM töltjük fel SEMMIVEL.** Jelen esetben a **Szerzo**-ben a **LISTÁT** tartalmazó property-t {**Konyvek** a property neve itt és az osztályban csak az **=null!** értéket adtuk meg neki! Ezt nemsokára profibban fogjuk csinálni!)

```
public List<Konyv> Konyvek { get; set; } = null!;
```

Míg az 1 ágon lévő osztályban azt a property-t, amely a másik osztály OBJEKTUMÁT tartalmazza (itt a Konyv osztályban a Szerzo property) **LÉTRE SEM HOZZUK A PÉLDÁNYOSÍTÁSKOR**!

Konkrétan így néz ki a dolog, ha a fájlok a *konyv.csv* és a *szerzo.csv* fájlokban vannak:

```
List<Konyv> konyvek = new List<Konyv>();
List<Szerzo> szerzok = new List<Szerzo>();
foreach (string i in File.ReadAllLines("szerzo.csv").Skip(1))
{
    string[] resz = i.Split(';');
    var szerzo = new Szerzo
    {
        Id = int.Parse(resz[0]),
        Nev = resz[1],
        Szulev = int.Parse(resz[2]),
        Halev = string.IsNullOrWhiteSpace(resz[3]) ? null : int.Parse(resz[3]),
        Nemzetiseg = resz[4],
        Konyvek = new List<Konyv>()
    };
    szerzok.Add(szerzo);
}
foreach (var i in File.ReadAllLines("konyv.csv").Skip(1))
{
    string[] resz = i.Split(';');
    var k = new Konyv
    {
        Id = int.Parse(resz[0]),
        Cim = resz[1],
        SzerzoId = int.Parse(resz[2]),
        Helyezes = int.Parse(resz[3])
    };
    konyvek.Add(k);
}
```

A *string.IsNullOrWhiteSpace(paraméter) IGAZ* értéket ad vissza, ha a paraméter üres, null vagy csak szóköz!

Ha kész a két objektumokból álló lista, akkor feltölthetjük a **Navigációs Property -et.**

Itt MANUÁLISAN TÖLTJÜK FEL a **Navigációs Property -et,** mert ebből lehet legjobban megérteni, hogy mi is történik a háttérben! **MEG KELL ÉRTENED ŐKET A TOVÁBBLÉPÉSHEZ, TEHÁT ADDIG NÉZD, AMÍG MEG NEM ÉRTED!!!** (fájlból történő feltöltésnél gyakran töltenek fel a valóságban is kézzel, teljes kontroll alatt tartjuk az eseményeket!)

Itt 3 kicsit különböző módszert mutatok. + Írd meg a programot is, hogy lásd!

Az első példában az **1:N** kapcsolat **1**-es irányából indulok (**INNEN A LEGEGYSZERŰBB**!!!) Végigmegyek a konyvek listán, ami a konyvek objektumokat tartalmazza és megkeresem a szerzo listában azt a szerzo objektumot, aki az adott könyvet írta. Ha megvan a szerző, akkor beállítom a **Konyv** osztály **Szerzo** propertyjének, a **Szerzo** osztály **Konyvek** propertyjéhez pedig (ami az adott szerző által írt könyveket tartalmazó **LISTA**) hozzáadom az adott **Konyv** osztályból származó obejektumot! (a könyv szerzője benne **VAN** a szerzők táblában ezért használhatunk **FIRST**-öt, nem fog **NULL** értékkel visszatérni soha!)

```
// a két lista összekapcsolása a Navigációs Property-k segítségével

foreach (var k in konyvek)
{
    var szerzo = szerzok.First(x => x.Id == k.SzerzoId);
    k.Szerzo = szerzo; // navigációs property beállítása a Konyv osztályban
    szerzo.Konyvek.Add(k); // navigációs property beállítása visszafelé a Szerzo osztályban
}
```

A második példa csak érdekesség, **NE HASZNÁLD, MERT LASSAN FUT**, de ettől még meg kell értened. Ebben az esetben az **N** ág felől indulunk. Végigmegyünk a szerzok listán és minden szerzőhöz végigszaladunk az összes könyvön. Ha találunk olyan könyvet, amit a szerző írt, akkor beteszzük a **Szerzo** osztály **Konyvek** propertyjébe (**LISTA**) és hozzáadjuk a **Konyv** osztály **Szerzo** propertyjéhez.

```
// a két lista összekapcsolása a Navigációs Property-k segítségével

foreach (var sz in szerzok)
{
    foreach (var k in konyvek)
    {
        if (k.SzerzoId == sz.Id)
        {
            sz.Konyvek.Add(k);
            k.Szerzo = sz;
        }
    }
}
```

A harmadik módszer a legjobb, messze ez fut a leggyorsabban, ami az adatbázisok esetén nagyon nem hátrány! Itt egy **szótárt** csinálunk a **Szerzo** osztályból. A szótár kulcsa az **Id** lesz a hozzá tartozó érték pedig maga a **Szerzo** objektum. Így nagyon gyorsan megtalálja minden könyvhöz a szerzőt a program!

```
// elkészítem a szótárt a szerzok listából (a kulcs az Id)
var szerzoDict = szerzok.ToDictionary(x => x.Id);
foreach (var k in konyvek)
{
    var szerzo = szerzoDict[k.SzerzoId];
    k.Szerzo = szerzo;
    szerzo.Konyvek.Add(k);
}
```

Végezetül már csak el kell menteni az adatokat a táblákba és készen is vagyunk. Itt arra kell figyelni, hogy a listákat tartományként tudjuk feltölteni. Utána jöhetnek a lekérdezések! 😊

```
// mentés az adatbázisba

using (var ctx = new KonyvDbContext())
{
    ctx.Szerzok.AddRange(szerzok);
    ctx.Konyvek.AddRange(konyvek);

    ctx.SaveChanges();
}
```