MS SQL ADATBÁZIS ÉS EF ALAPOK

Segédlet a Webportálok fejlesztése c. tárgyhoz

Gincsai Gábor

2025.

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

1117 Budapest, Magyar tudósok körútja 2.
Q. ép. B. szárny 207. • www.aut.bme.hu
Telefon: 463-2870 • Fax: 463-2871

BEVEZETÉS

CÉLKITŰZÉS

A labor célja, hogy átismételjük a MS SQL Server ismereteket, illetve azon hallgatóknak akik még nem foglalkoztak adatbázisokkal a gyakorlatban megmutassuk, hogyan lehet táblákat létrehozni, lekérdezéseket és tárolt eljárásokat készíteni.

A labor során a hallgatók egy-egy példán keresztül ismerkednek meg a Táblalétrehozást, adatbázis lekérdezések és módosítások készítését, valamint a tárolt eljárások írását.

ELŐFELTÉTELEK

A labor elvégzéséhez szükséges eszközök:

- MS SQL Server 2016, vagy újabb
- Microsoft SQL Managenet Studio 18
- Visual Studio 2022
- .NET 8
- Az sql projecthez .NET Framework 4.8.1 (igen Framework ami még mindig támogatott)

Letölthető fájlok a laborhoz

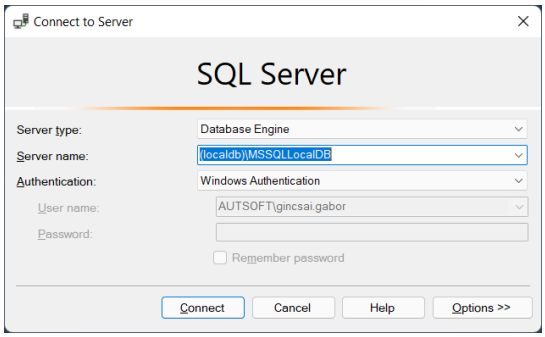
- testData.zip

AMIT ÉRDEMES ÁTNÉZNED

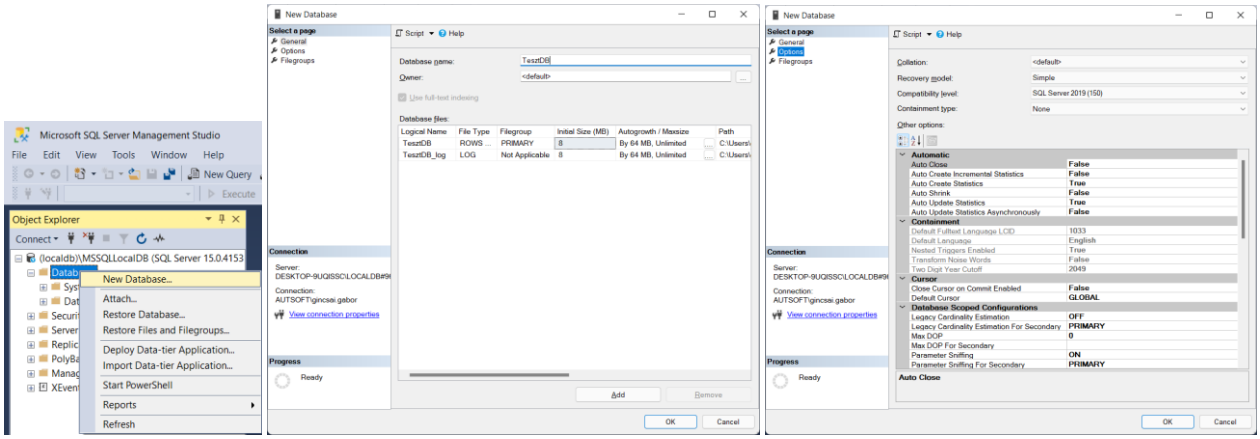
- Adatbázis kezelési alapok

ADATBÁZIS LÉTREHOZÁSA

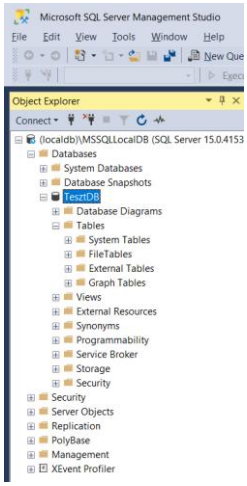
1. Indítsuk el a Microsoft SQL Server Management Studiot-t, majd csatlakozzunk a lokális adatbázis szerverhez, amit a (localdb)\MSSQLLocalDB névvel érünk el.
   - A lokális adatbázis szerver nem teljes SQL szerver és tipikusan fejlesztéshez érdemes használni. A Visual Studio alapértelmezés szerint feltelepíti.
   - Ha a „nagy” SQL szerverhez szeretnénk kapcsolódni, ami a lokális gépen fut, akkor a . (pontot) adjuk meg mint szervernév. Ezt külön telepíteni kell.



2. Az Object explorerben jobb klikk a Databases-re és válasszuk a New Database opciót, majd adjuk meg az adatbázis nevét, ami legyen **TesztDB**. Bal oldalon még az Options fül lehet érdekes, ahol számos adatbázis beállítást meg lehet adni, de ezeket most nem módosítjuk, jó az alapértelmezett beállítás.

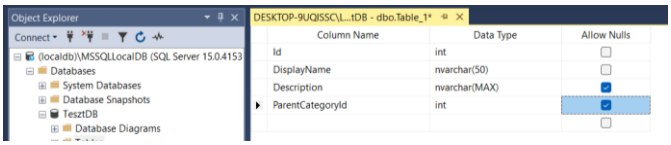


3. Az Ok-ra kattintva létrejön az adatbázis és az Object Explorerben a Databases alatt meg is jelenik.

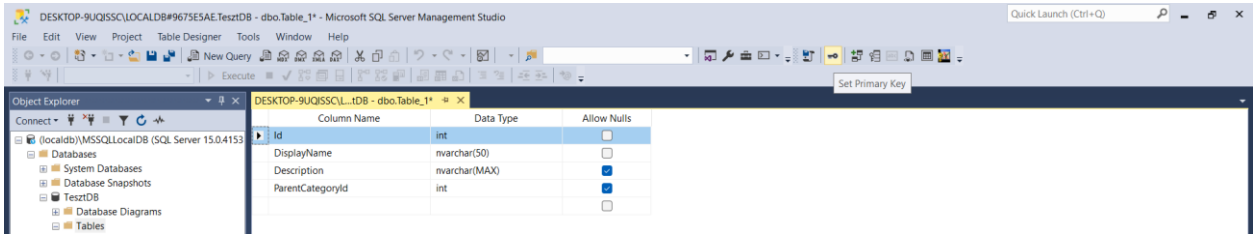


PRODUCT ÉS CATEGORY TÁBLA LÉTREHOZÁSA MANAGEMENT STUDIOVAL

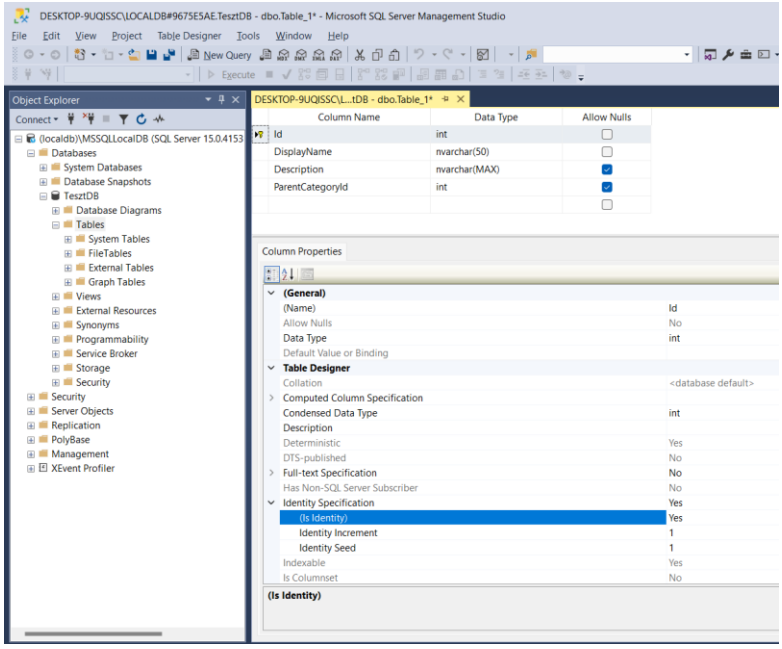
1. Nyissuk ki az Object Explorerben a TestDB-t, majd jobb gomb a Tables >> New Table.
2. Majd adjuk meg, hogy milyen oszlopai lesznek a táblának, illetve az oszlopok típusait. Először a kategória táblát hozzuk létre.



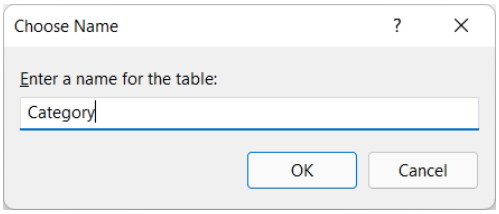
3. Válasszuk ki az ID-t, majd állítsuk be, hogy az legyen az elsődleges kulcs. Ehhez fent a menüben kattintsunk a kulcs ikonra. Ha jól csináltuk az ID előtt megjelenik egy kulcs.



4. Állítsuk be, hogy az elsődleges kulcsot automatikusan ossza az SQL server. Ehhez válasszuk ki az ID-t majd a tulajdonságai között nyissuk le az Identity Specification-t és állítsuk az (Is Identity) értékét true-ra. A Seed a kezdő sorszámot adja meg az increment pedig azt, hogy mennyivel növelje az ID-kat.
   Pl. Ha a seed 3 és az increment 2 akkor az ID-k rendre 3, 5, 7, 9, 11 lesznek.

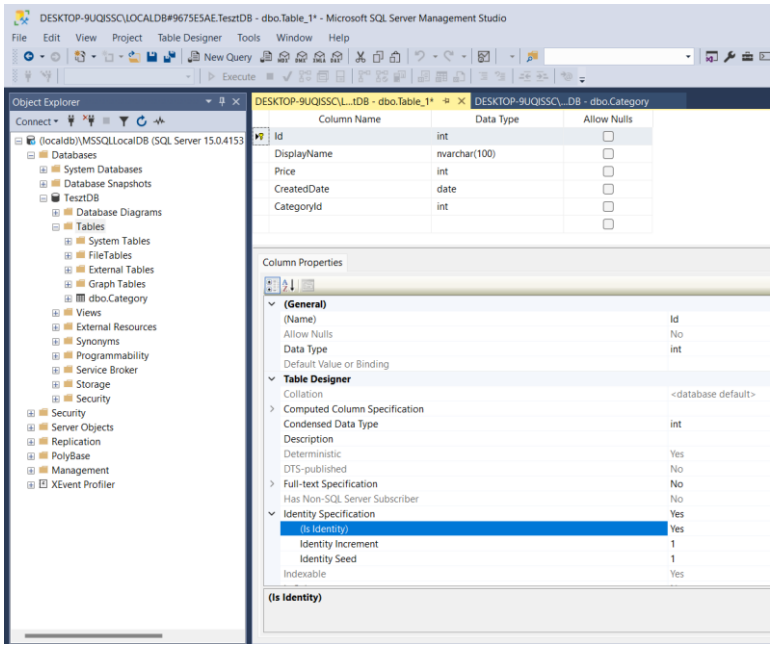


5. Mentsük el a táblánkat Ctrl+S segítségével. Ekkor fog rákérdezni a tábla nevére is.

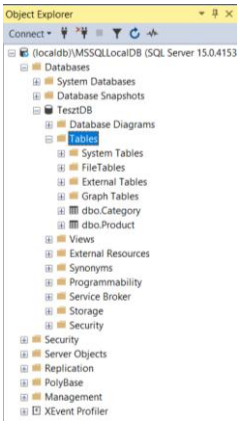


Ha nem jelenik meg a mentés után a tábla az Object exprorerben akkor frissítsük az adatbázist TesztDB → Jobb klikk → Refresh

6. Hozzuk létre ugyanígy a Product táblát is.

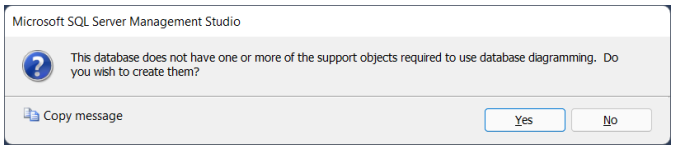


7. Ha mindent jól csináltunk az Object explorerben az alábbit kell látnunk.

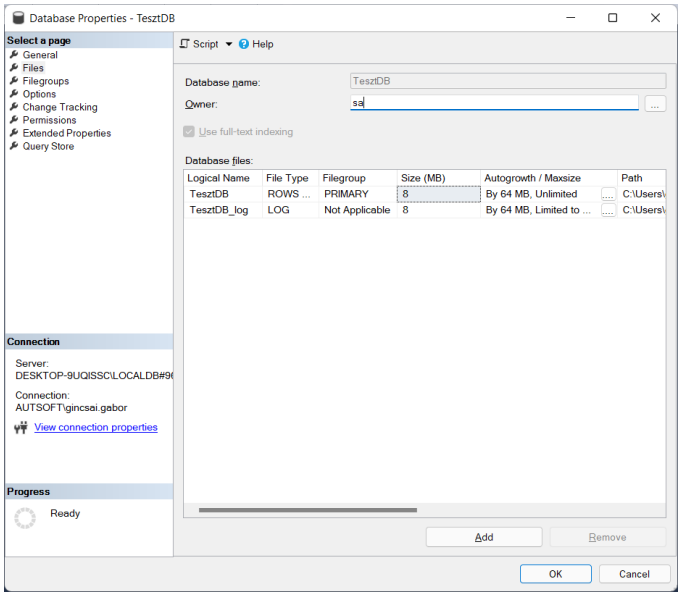


KÉSZÍTSÜNK ADATBÁZIS DIAGRAMOT

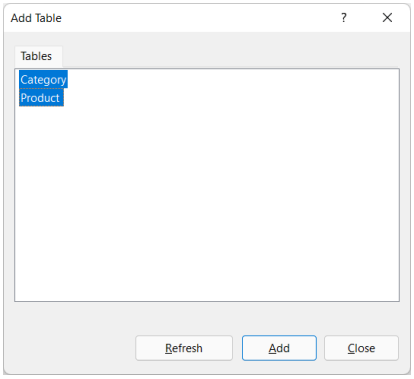
1. Adatbázis diagram készítéséhez nyissuk ki az Object explorerben a TeszDB alatti Database Diagrams részt.
2. A felbukkanó ablakban kattintsunk az OK-ra, hogy valóban szeretnénk létrehozni diagramot.



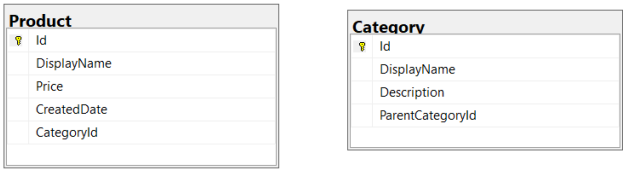
Ha nem a fenti képernyőt kapjuk, hanem egy hibaüzenetet, akkor az egy ismert bug miatt van. Nem tudja úgy létehozni az adatbázis diagramot, ha az adatbázis tulajdonosa nem az **sa** felhasználó. Ahhoz, hogy az sa legyen az owner, kattintsunk az TesztDB-n jobb gombbal, majd válasszuk a properties opciót. Ezt követően a bal oldalon a Files menüpontot választva tudjuk megváltoztatni az adatbázis tulajdonosát. Egyszerűen az Owner mögé írjuk be, hogy sa és kattintsunk az OK-ra. Ezt követően már létre tudjuk hozni az adatbázis diagrammot.



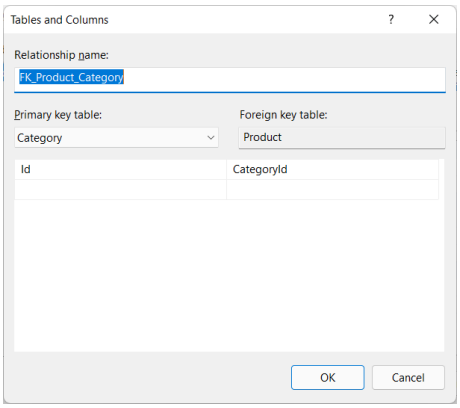
3. Majd jobb gomb a Database Diagrams-on és válasszuk a New Database diagram-ot, majd az előugró ablakban válasszuk ki mindkét táblát, majd Add és Close. Itt adjuk meg mely táblák kerüljenek rá a diagramra.



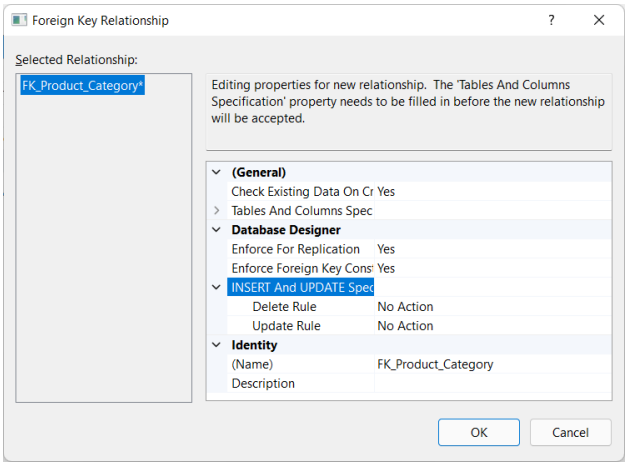
4. Ekkor az alábbi diagramot készíti el a Management Studio



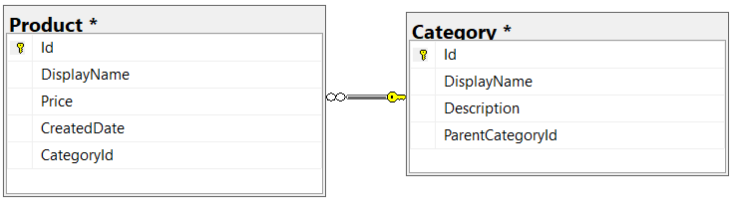
5. Mint látható az ábrán a Product táblában szerepel a CategoryId, de az nem külső kulcs. Ahhoz, hogy ezt meg tudjuk adni, kattintsunk a Product tábla CategoryId sorára, majd miközben lenyomva tartjuk a gombot húzzuk az egeret a Category tábla ID oszlopára. Ekkor az alábbi ablak fog előjönni.



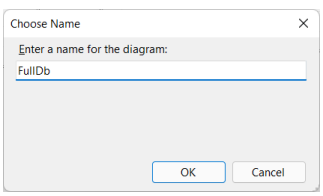
6. Ellenőrizzük, hogy valóban a Category tábla ID oszlopa a Primary key és a Product tábla CategoryId oszlopa a Foreign key. Majd kattintsunk az OK-ra. Az ezt követő ablakban a külső kulcshoz tartozó legfontosabb tulajdonságokat lehet megadni. Pl: Cascade törlés, vagy a kulcs neve, ami jelen esetben FK_Product_Category.

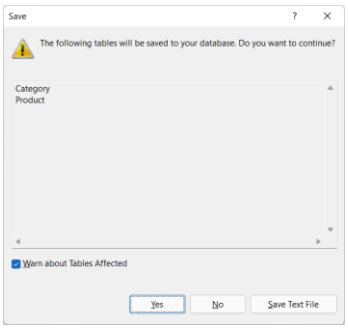


7. Az OK-al zárjuk be ezt az ablakot is, és ekkor a diagramon megjelenik a külső kulcs is.

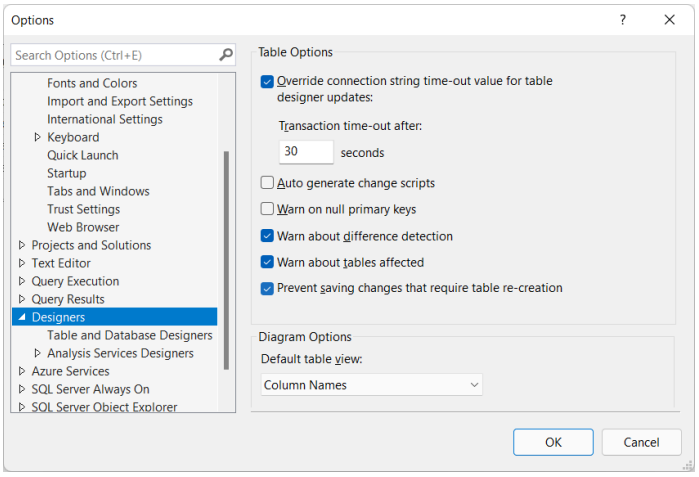


8. Majd mentsük el a diagramot, ami a táblákat is fogja módosítani (mert csillag van a tábla nevek mellett) Ctrl + S, majd adjuk meg a diagram nevét és hagyjuk jóvá a táblamódosításokat a Yes-el.



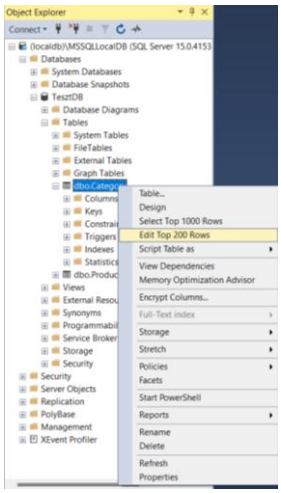


9. Ha nem engedi elmenteni a módosításokat, az azért van, mert a Designerben le van tiltva az összes olyan módosítás, amihez újra létre kellene hozni a táblát. (Adatvédelmi okok.) Ezt a Tools >> Option menüben lehet kikapcsolni, ahogy az az alábbi ábrán is látható.

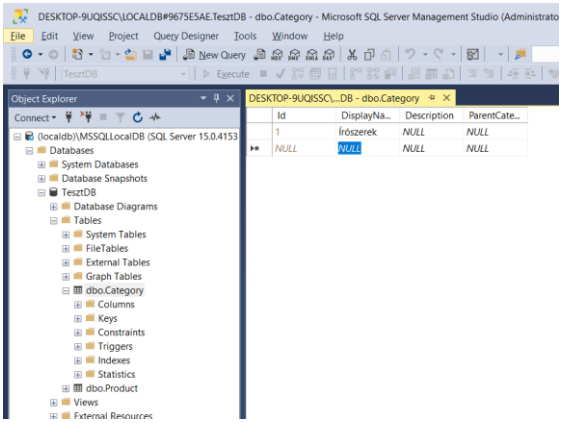


TESZT ADATOK FELVITELE GRAFIKUS FELÜLETEN MANAGEMENT STUDIOVAL

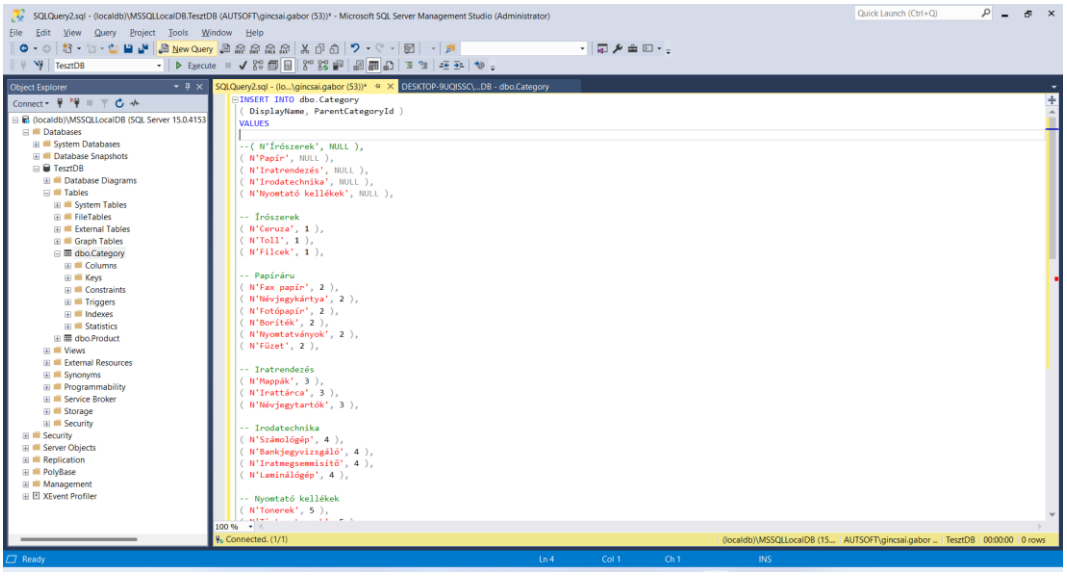
1. Az object exprorerben a Category táblán jobb klikk >> Edit Top 200 Rows segítségével lehetőségünk van kézzel táblázatos formában megadni a teszt adatokat.



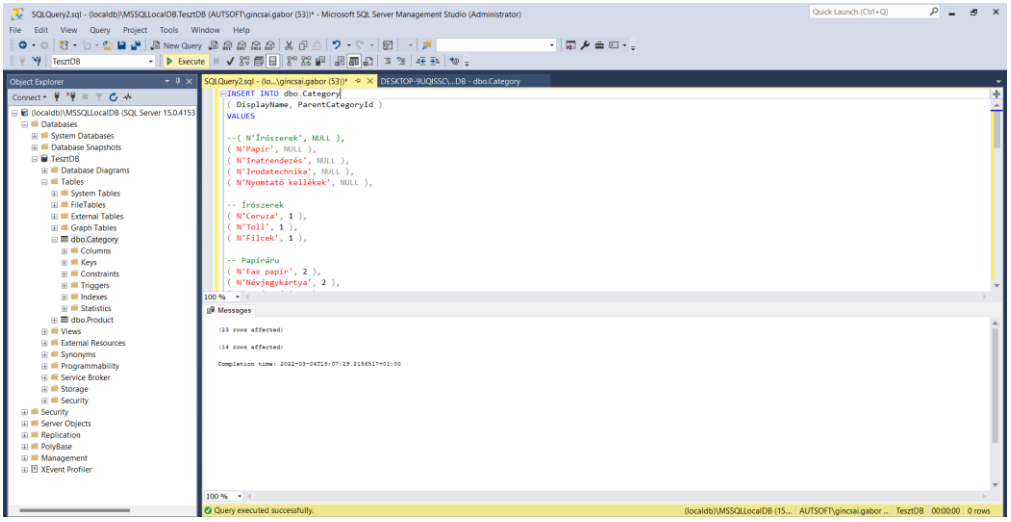
2. Adjunk hozzá egy kategóriát a megjelenő táblázathoz. A sort akkor menti el, ha elnavigálunk róla. Fontos, hogy mivel az ID-t az adatbázis generálja ezért az nem szabad megadni!



3. A többi adatot szkript segítségével töltjük be. Ehhez kattintsunk a **New Query** ikonra a felső sorban, majd másoljuk át a tesztData.sql tartalmát az editor ablakba. Mivel már az Írószereket beszúrtuk ezért az kommentezzük ki. SQL-ben a kommentet -- karakterekkel lehet készíteni. Figyeljünk rá, hogy az ID-at az adatbázis osztja növekvő sorrenben egyesével, de akkor is kioszt egy ID-t ha sikertelen a beszúrás. Tehát ellenőrizzük, hogy az Írószerek tényleg az 1-es ID-t kapta-e. Ha nem akkor a szkriptben módosítani kell a fix értékeket, hogy helyesek legyenek az adatok.



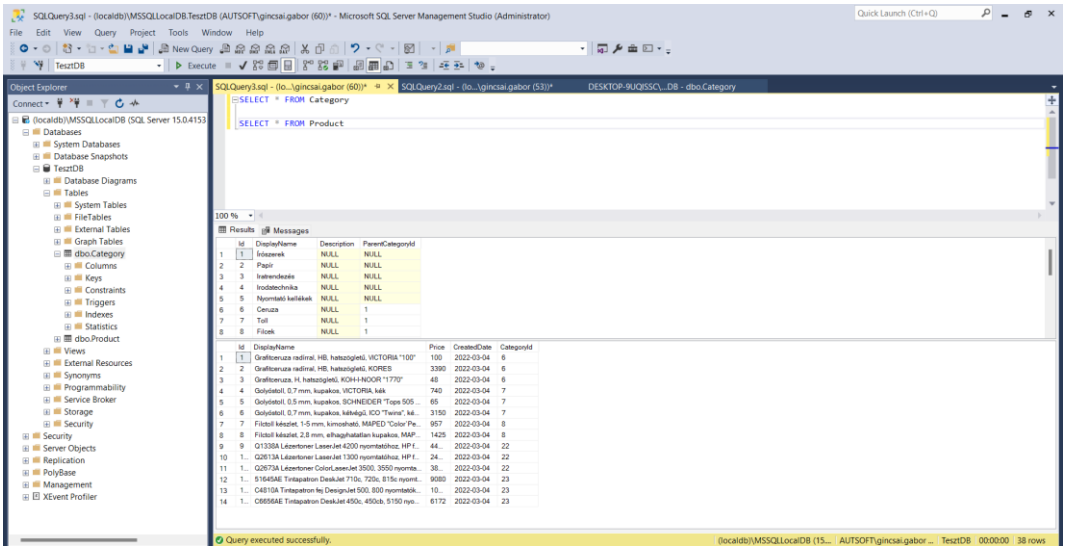
4. Ha megva a script az Execute ikonnal tudjuk lefuttatni. A futás eredményét a lent megjelenő Messages ablakban látjuk.



5. Készítsünk egy lekérdezést, amivel meggyőződünk róla, hogy tényleg minden kategóriát és terméket sikerült beszúrni. A két lekérdezés amit meg kell hozzá írni mindent lekérdez a Category és Product táblákból, az alábbi lekérdezéssel:

```
SELECT * FROM Category
SELECT * FROM Product
```

Kattintsunk az Execute ikonra, majd a Results ablakban láthatjuk egymás alatt a két tábla adatait.



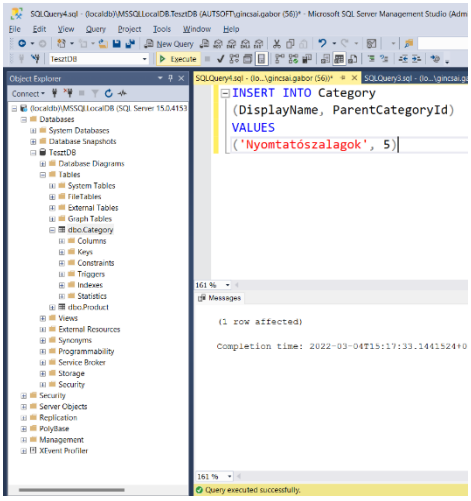
ADATBÁZIS LEKÉRDEZÉSEK

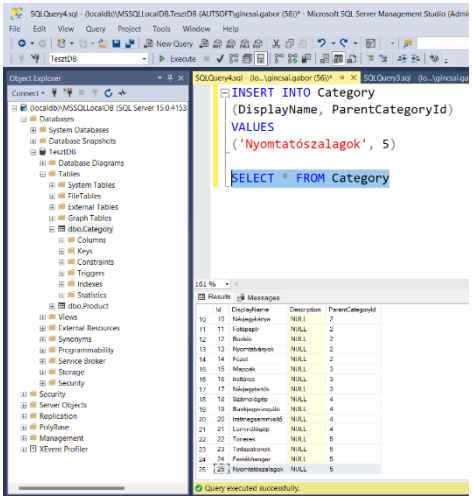
ÚJ KATEGÓRIA BESZÚRÁSA

1. Készítsünk egy query-t ami beszúr egy új kategóriát.

```
-- Új kategória beszúrása
INSERT INTO Category
(DisplayName, ParentCategoryId )
VALUES
('Nyomtatószalagok', 5)
```

2. Futtassuk le, majd kérdezzük le az összes kategóriát, hogy valóban sikerült-e beszúrni. Az összes kategória lekérdezésénél csak a kijelölt sor futtattuk.



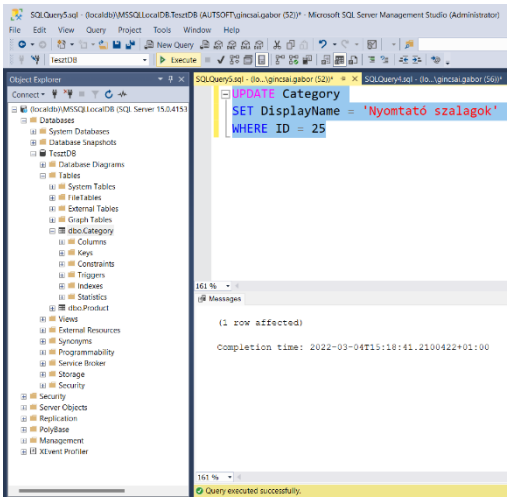


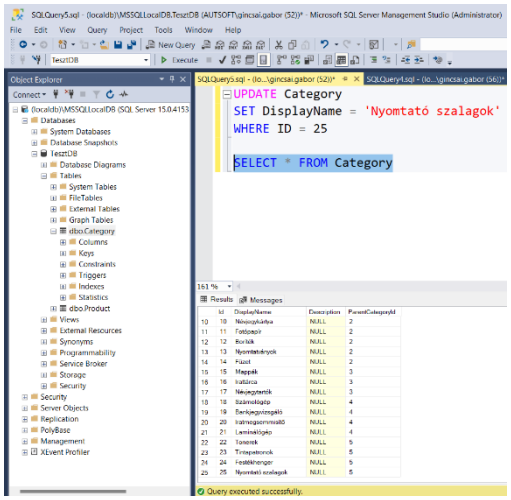
KATEGÓRIA MÓDOSÍTÁSA

1. Készítsünk egy lekérdezést, ami módosítja a beszúrt sort. (Jelenleg ID = 25)

```
-- Kategória módosítása
UPDATE Category
SET DisplayName = 'Nyomtató szalagok'
WHERE Id = 25
```

2. Futtassuk le, majd kérdezzük le az összes kategóriát, hogy valóban sikerült-e módosítani az adatokat. Az összes kategória lekérdezésénél csak a kijelölt sort futtatuk.



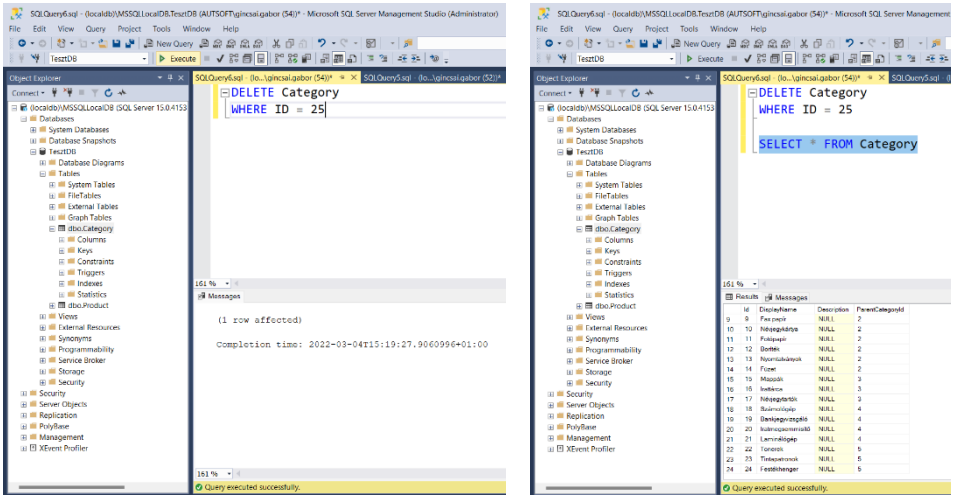


KATEGÓRIA TÖRLÉSE

1. Készítsünk query-t ami törli az általunk létrehozott kategóriát (ID = 25)

```
-- Kategória törlése
DELETE Category
WHERE ID = 25
```

2. Futtassuk le, majd kérdezzük le az összes kategóriát, hogy valóban sikerült-e törölni a kívánt rekordot.

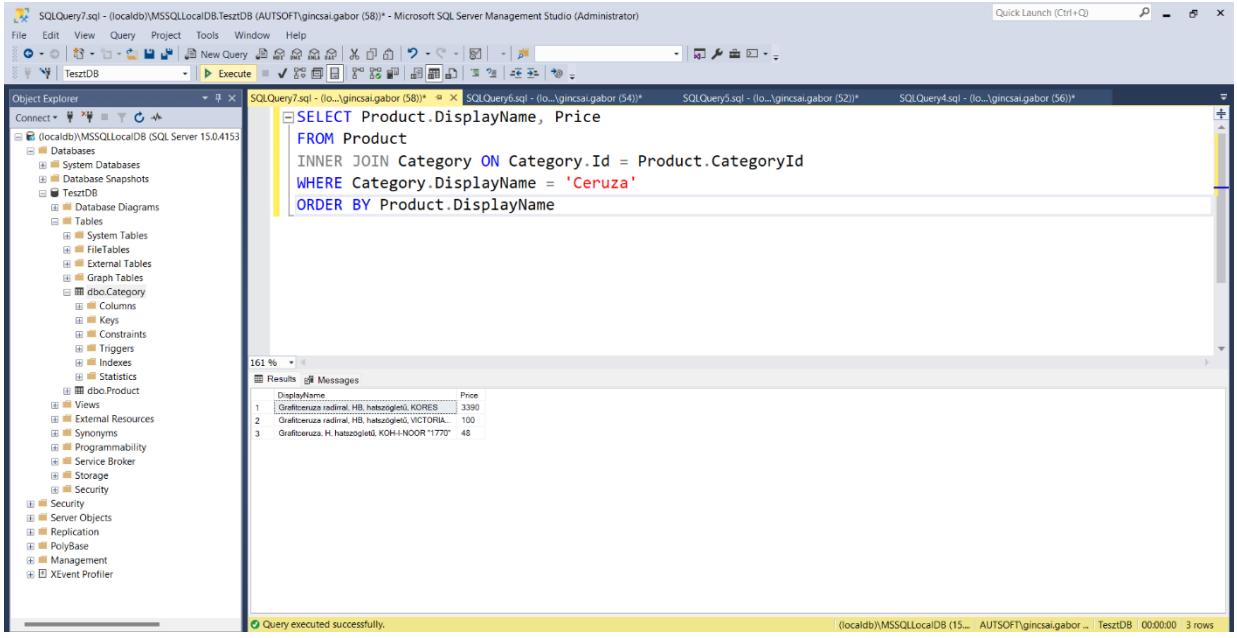


ADOTT KATEGÓRIÁBA TARTOZÓ ÖSSZES TERMÉK LEKÉRDEZÉSE, ABC SORRENDBEN.

1. Készítsünk egy összetettebb lekérdezést is, ami visszaadja az adott kategóriába tartozó összes termék nevét és árát.

```
SELECT Product.DisplayName, Price FROM Product
INNER JOIN Category ON Category.ID = Product.CategoryId
WHERE Category.ID = 6
```

2. Futtassuk le és nézzük meg mit ad vissza.



TÁROLT ELJÁRÁS KÉSZÍTÉSE

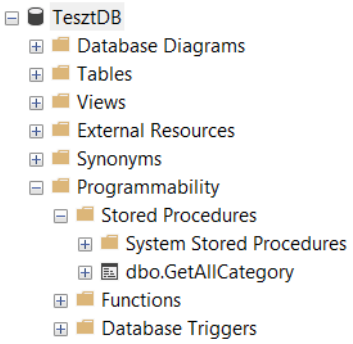
A korábban elkészített lekérdezésekből (összes kategória lekérdezés, kategória beszúrása, módosítása, törlése és a kategória termékeinek lekérdezéséből) készítsünk tárolt eljárást.

GETALLCATEGORY

1. Összes kategória lekérdezés

```
CREATE PROCEDURE GetAllCategory
AS

SELECT * FROM Category
```

2. Ha ezt elkészítettük futtassuk is le, amire azt a választ kapjuk, hogy sikeresen lefutott, de nem adja vissza a kategóriákat. Ez azért van, mert ilyenkor egy tárolt eljárást hoz létre, amit később futtatni tudunk.
3. Az *Object Explorerben* is meg tudjuk nézni, hogy sikeresen létrehozta a GetAllCategory nevű tárolt eljárást. (*TestDB >> Programmability >> Stored Procedures* alatt)



4. Futtassuk az elkészített tárolt eljárásunkat, és nézzük meg az eredmény.

```
EXEC GetAllCategory
```

Results Messages

	Id	DisplayName	Description	ParentCategoryId
1	1	Írószerek	NULL	NULL
2	2	Papír	NULL	NULL
3	3	Iratrendezés	NULL	NULL
4	4	Irodatechnika	NULL	NULL
5	5	Nyomtató kellékek	NULL	NULL
6	6	Ceruza	NULL	1
7	7	Toll	NULL	1
8	8	Filcek	NULL	1
9	9	Fax papír	NULL	2
10	10	Névjegykártya	NULL	2
11	11	Fotópapír	NULL	2
12	12	Boríték	NULL	2
13	13	Nyomtatványok	NULL	2
14	14	Füzet	NULL	2
15	15	Mappák	NULL	3
16	16	Irattárca	NULL	3
17	17	Névjegytartók	NULL	3
18	18	Számológép	NULL	4
19	19	Bankjegyvizsgáló	NULL	4
20	20	Iratmegsemmisítő	NULL	4
21	21	Laminálógép	NULL	4
22	22	Tonerek	NULL	5
23	23	Tintapatronok	NULL	5
24	24	Festékhenger	NULL	5

INSERTCATEGORY

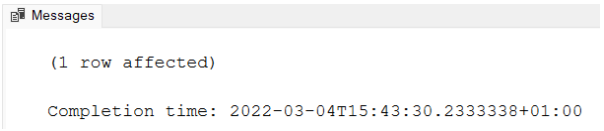
1. Hozzunk létre egy olyan tárolt eljárást ami létrehoz egy új kategóriát. Két bemenő paramétere van a kategória neve és a szülő kategória azonosítója.

```
-- Új kategória létrehozása
CREATE PROCEDURE InsertCatgory
      @DisplayName NVARCHAR(100),
      @ParentCategoryId INT
AS

INSERT INTO Category
(DisplayName, ParentCategoryId )
VALUES
(@DisplayName, @ParentCategoryId )
```

2. Futtassuk a tárolt eljárást. Ne feledkezzünk meg a paraméterekről sem.

```
-- Futassuk a tárolt eljárást
EXEC InsertCatgory 'Nyomtatószalagok', 5
```

3. Nézzük meg az eredményt.



4. Ha lekérdezzük az összes kategóriát a GetAllCategory tárolt eljárással, akkor láthatjuk, hogy létrejött a kívánt kategória. Figyeljük meg, hogy az ID-ja 26 lett.

Results Messages

	Id	DisplayName	Description	ParentCategoryId
18	18	Számológép	NULL	4
19	19	Bankjegyvizsgáló	NULL	4
20	20	Iratmegsemmisítő	NULL	4
21	21	Laminálógép	NULL	4
22	22	Tonerek	NULL	5
23	23	Tintapatronok	NULL	5
24	24	Festékhenger	NULL	5
25	26	Nyomtatószalagok	NULL	5

Azért 26 az ID, mert korábban már beszúrtunk egy kategóriát, ami a 25-öst ID-t kapta, ami később töröltük, viszont az adatbázisszerver még egyszer nem osztja ki a már egyszer kiosztott ID-t. Akkor sem, ha már töröltük.

UPDATECATEGORY

1. Hozzunk létre egy olyan tárolt eljárást ami módosít egy kategóriát. Három bemenő paramétere van. A módosítandó kategória azonosítója és a kategória új neve és az új szülő kategóriája.

```
-- Kategória módosítása
CREATE PROCEDURE UpdateCategory
      @ID INT,
      @DisplayName NVARCHAR(100),
      @ParentCategoryId INT
AS

UPDATE Category
SET DisplayName = @DisplayName,
      ParentCategoryId = @ParentCategoryId
WHERE
      ID = @ID
```

2. Futtassuk a tárolt eljárást. Ne feledkezzünk meg a paraméterekről sem.

```
-- Tárolt eljárás futtatáts
EXEC UpdateCategory 26, 'Nyomtató szalagok', 5
```

3. Nézzük meg az eredményt.

4. Ha lekérdezzük az összes kategóriát a GetAllCategory tárolt eljárással, akkor láthatjuk, hogy módosult a kívánt kategória.

Results Messages

	Id	DisplayName	Description	ParentCategoryId
20	20	Iratmegsemmisítő	NULL	4
21	21	Laminálógép	NULL	4
22	22	Tonerek	NULL	5
23	23	Tintapatronok	NULL	5
24	24	Festékhenger	NULL	5
25	26	Nyomtató szalag...	NULL	5

DeleteCategory

1. Hozzunk létre egy olyan tárolt eljárást, ami törli a megadott kategóriát.

```
-- Kategória tötlése
CREATE PROCEDURE DeleteCategory
      @ID INT
AS

DELETE Category
WHERE
      ID = @ID
```

2. Futtassuk a tárolt eljárást. Ne feledkezzünk meg a paraméterekről sem.

```
-- Tárolt eljárás futtatáts
EXEC DeleteCategory 26
```

3. Nézzük meg az eredményt.

4. Ha lekérdezzük az összes kategóriát a GetAllCategory tárolt eljárással, akkor láthatjuk, hogy a kívánt kategóriát sikerült törölni.

Results Messages

	Id	DisplayName	Description	ParentCategoryId
18	18	Számológép	NULL	4
19	19	Bankjegyvizsgáló	NULL	4
20	20	Iratmegsemmisítő	NULL	4
21	21	Laminálógép	NULL	4
22	22	Tonerek	NULL	5
23	23	Tintapatronok	NULL	5
24	24	Festékhenger	NULL	5

GetProductsInCategory

1. Hozzunk létre egy olyan tárolt eljárást, ami visszaadja az adott kategóriában lévő termékek nevét és árát.

```
-- Kategóriában lévõ termékek lekérdezése
CREATE PROCEDURE GetProductsInCategory
      @CategoryId INT
AS

SELECT Product.DisplayName, Price
FROM Product
INNER JOIN Category ON Category.ID = Product.CategoryId
WHERE
      Category.ID = @CategoryId
```

2. Futtassuk a tárolt eljárást. Ne feledkezzünk meg a paraméterekről sem.

```
-- Tárolt eljárás futtatása
EXEC GetProductsInCategory 6
```

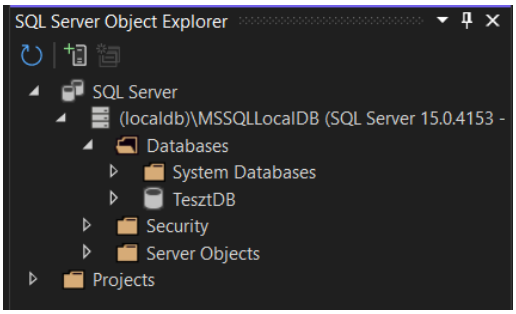
3. Nézzük meg az eredményt.

Results | Messages

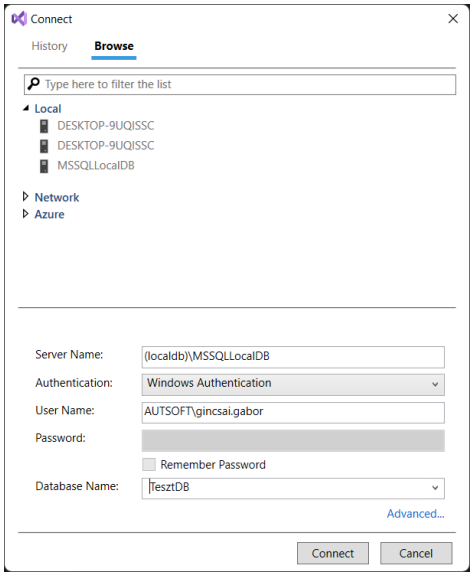
	DisplayName	Price
1	Grafitceruza radírral, HB, hatszögletű, VICTORIA...	100
2	Grafitceruza radírral, HB, hatszögletű, KORES	3390
3	Grafitceruza, H, hatszögletű, KOH-I-NOOR "1770"	48

VISUAL STUDIOBÓL IS NÉZZÜK MEG A SERVER EXPLORER LEHETŐSÉGEIT.

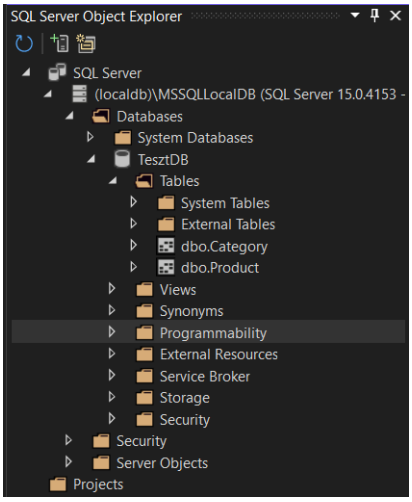
1. Indítsuk el a Visual Studio-t, majd válasszuk a Continue without code-ot. Ezt követően az SQL Server Object Explorer ablakban kapcsolódjuk a létrehozott adatbázisunkhoz (TesztDB)

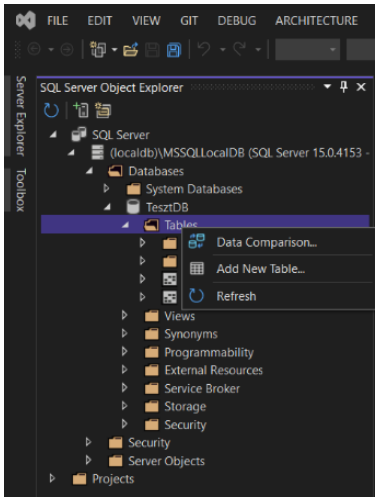


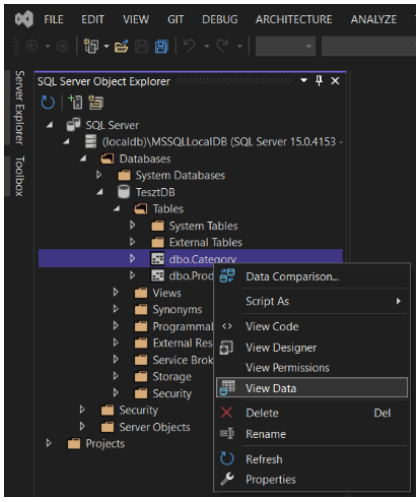
2. Ha nem látunk egy adatbázist sem, akkor kattintunk fent az Add SQL Server ikonra, majd a felugró ablakban tallózzuk ki az adatbázist.



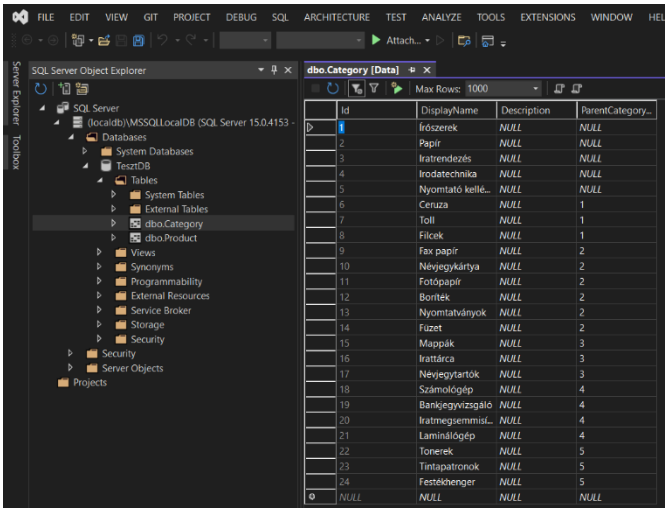
3. Ezt követően már meg tudjuk nézni az adatbázisban lévő táblákat és tárolt eljárásokat is. A Tables-re kattintva jobb gombbal vehetünk fel új táblákat, vagy egy táblára kattintva pl: dbo.Category meg tudjuk nézni a benne lévő adatokat



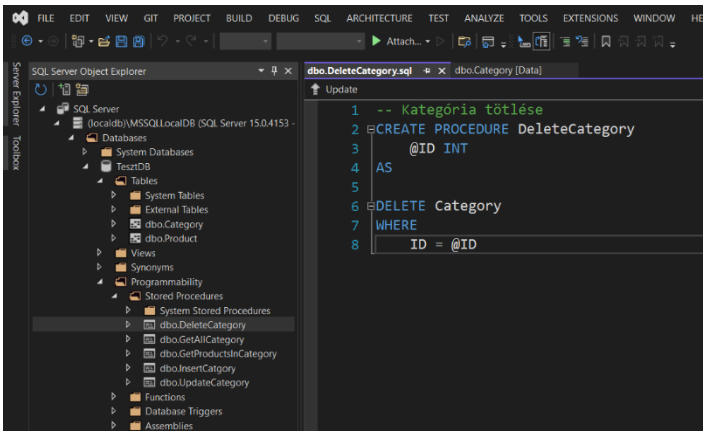




4. Nézzük meg, hogy a Categories táblában milyen adatokat látunk. Ehhez a View Data-ra kell kattintani.

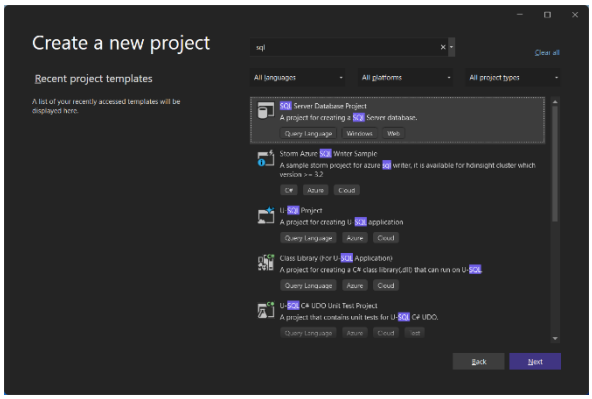


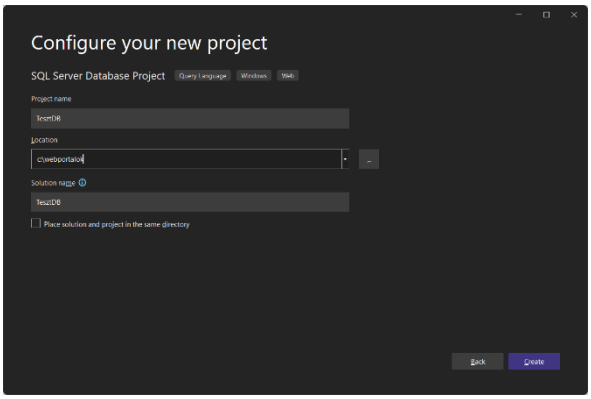
5. Módosíthatjuk a tárolt eljárásokat is a tárolt eljárás nevén duplán kattintva.



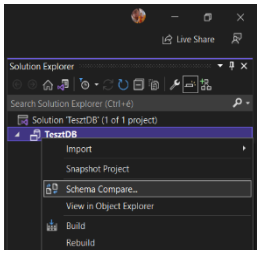
VISTUAL STUDIO DB PROJECT LÉTREHOZÁSA ÉS SÉMA COMPARE-EL FRISSÍTENI.

1. Hozzunk létre egy új projectet a Visual Studioban amiben az adatbázisunkat szeretnénk majd szkript formájában tárolni. Ha több ember dolgozik egy projekten, akkor az adatbázis nem megfelelő kezelése sok problémát tud felvetni. Mindenkinél a megfelelő verzió legyen, új tesztadatok generálása, DB újrahúzása egyszerű legyen, tesztadatok ne vesszenek el... Ezekre ad kiváló megoldást az *Sql Server Database Project*.
   Ahhoz, hogy ezt a projekttípus elérjük, telepíteni kell a Visual Studio 2022 intallerben az *SQL Server Data Tools*-t.[1]

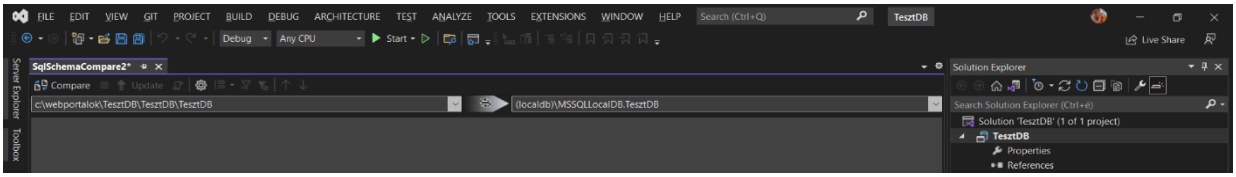




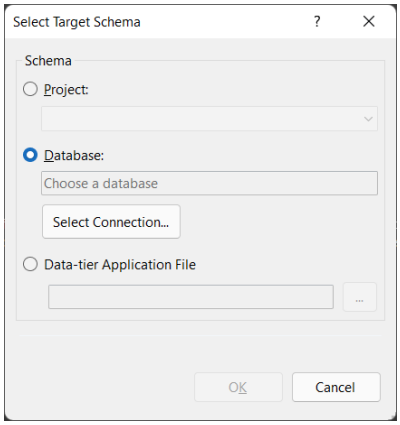
2. Ahogy a *Solution Exprorerben* is látszik, a projektünk még teljesen üres. A már létrehozott adatbázisunk sémáját importáljuk be ebbe a projectbe. Ehhez a TezstDB projekten jobb klikk >> *Compare Schema* opciót válasszuk.

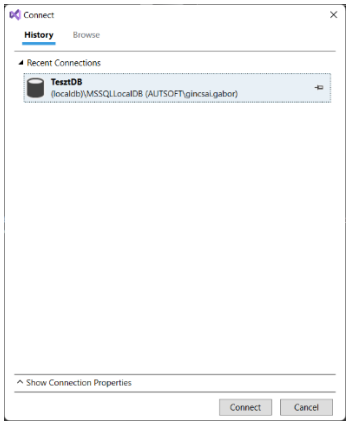


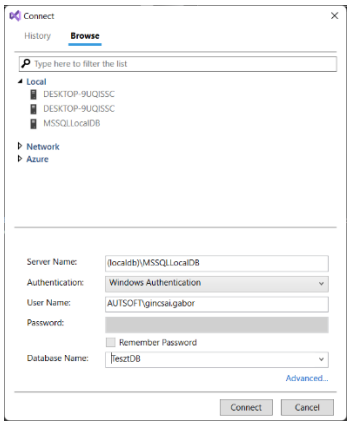
3. A megnyíló ablakban felül adjuk meg, hogy mit és mivel szeretnénk összehasonlítani.



Ha nem ajánlja fel a *Visual Studio* a TesztDB-t akkor vegyük fel új kapcsolatként, vagy az előzményekből vagy mint egy új kapcsolat.

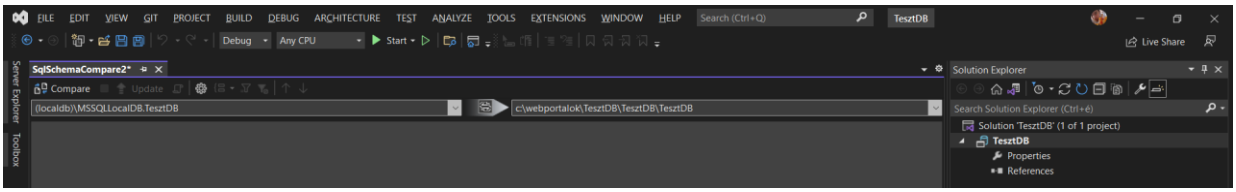




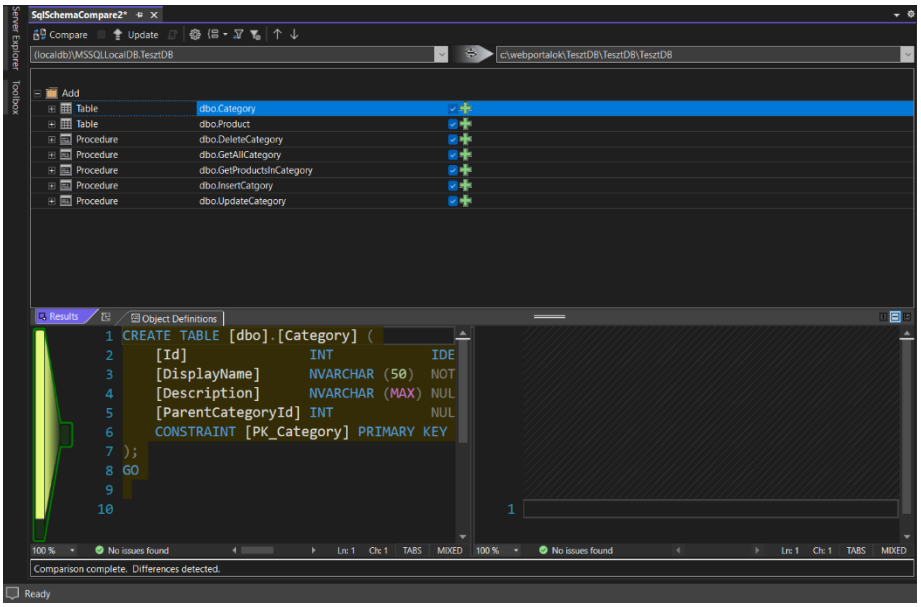


[1] https://docs.microsoft.com/en-us/sql/ssdt/download-sql-server-data-tools-ssdt?view=sql-server-ver15

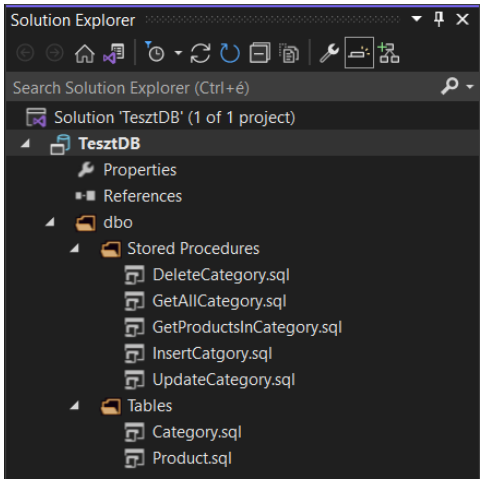
4. Fontos arra is figyelni, hogy a bal oldalt a forrás és jobb oldalt a cél szerepel. Tehát nekünk pont meg kell fordítani a két elemet, amit a közöttük lévő nyilakkal tehetünk meg. Így az adatbázis alapján fogja frissíteni a projektet.



5. Ezt követően, ha a *Compare*-re kattintunk kapunk egy listát, hogy a két séma miben tér el. Itt adhatjuk meg, hogy mi kerüljön módosításra és mi ne.

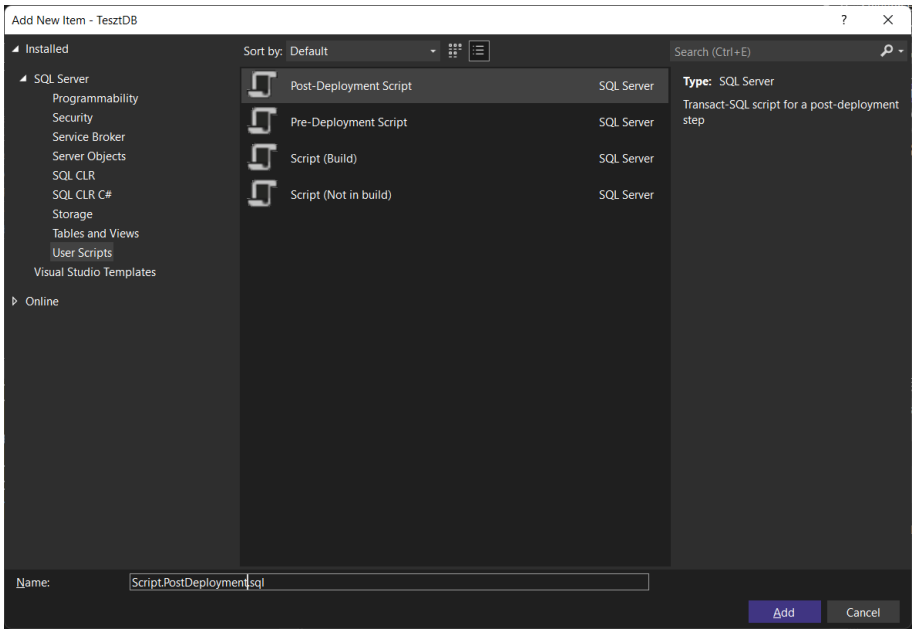


6. Kattintsunk az *Update* ikonra fent, hogy a módosításokat végrehajtsa. Megerősítés után a módosításokat elvégzi és már az adatbázis projectben lesznek a táblák és tárolt eljárások is. (Mind a *Schema Viewban*, mind pedig a *Solution Explorerben*.)



TESZTADATATOK LÉTREHOZÁSA SZRIPTBŐL.

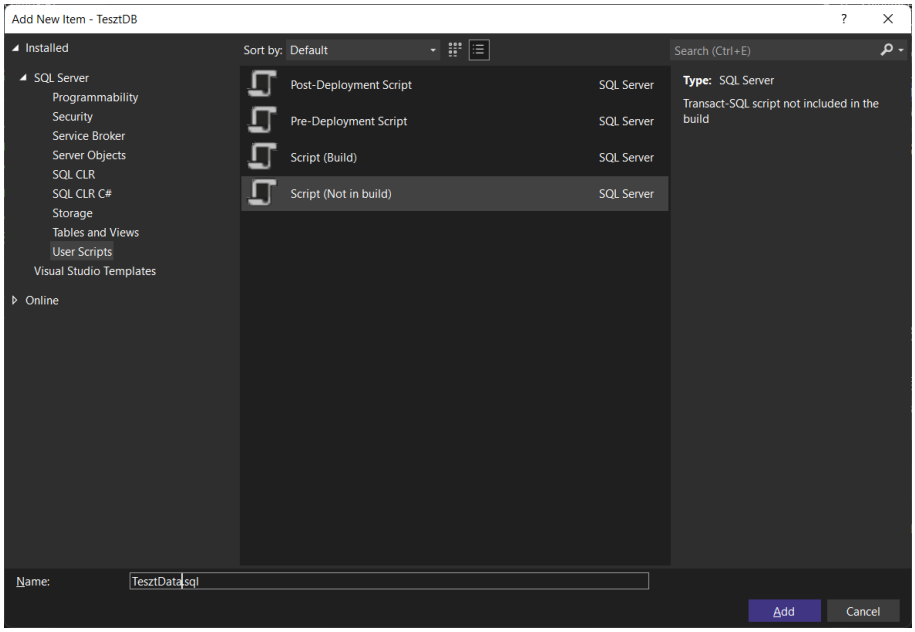
1. Már csak azt kell megoldani, hogy teszt adatok is kerüljenek az adatbázis szkriptek köze, amit egy kattintással tudunk majd deployolni. Ehhez kattintsunk jobb gombbal a projektre és *Add New Item*, ahol a *UserScripts* kategóriából válasszuk a *Post-Deployment Script*-et.



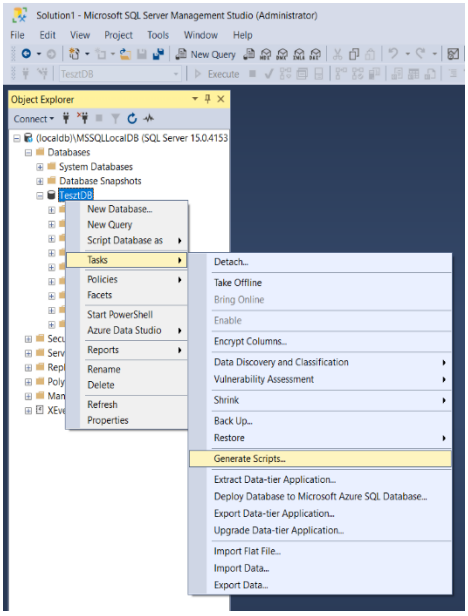
2. Ebbe a fájlba vegyük fel, hogy deployment után futtassa le a TestData.sql-t, amit nemsokára elkészítünk.

```
:r TesztData.sql
```

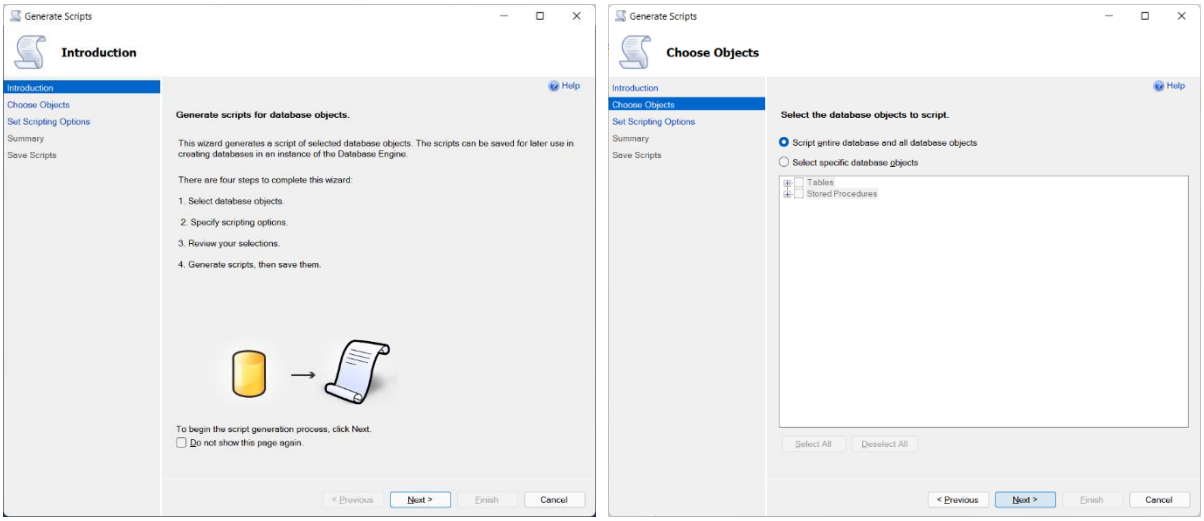
3. Hozzuk létre a TesztData.sql-t a Post Deployment könyvtárban.



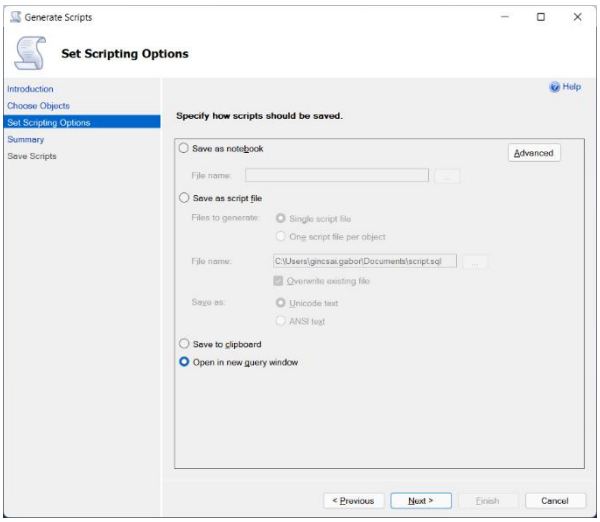
4. Mivel az adatbázisban bár benne vannak ezek a sorok onnan kellene átemelni. A Schema compernél azért nem jöttek át, mert az adatok nem a séma részei.
5. Az adatokat az SQL Managemt Studióban az TestDB-n jobb klikk → Tasks → Generate Scripts... segítségével tudjuk exportálni egy szkriptbe.

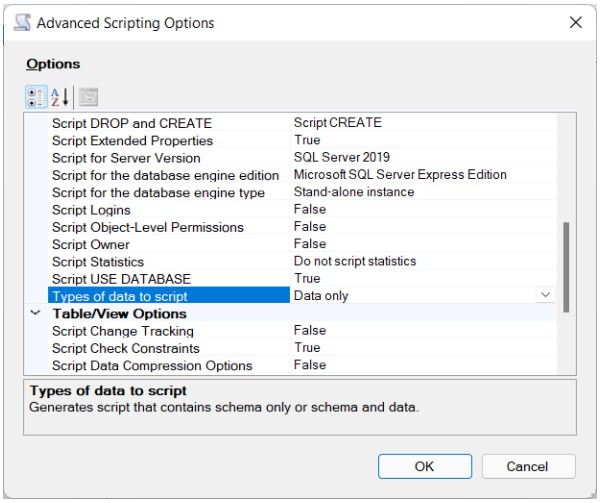


6. A Generate Scripts... egy varázslót indít el. Az első képernyőn üdvözöl minket, majd Next-el továbblépve kiválaszthatjuk, hogy mit szeretnénk exportálni. Az alapértelmezett beállítást, hogy mindent pont megfelel nekünk.

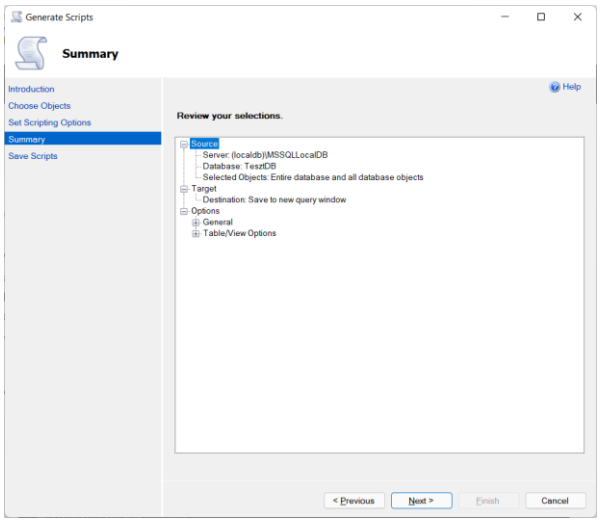


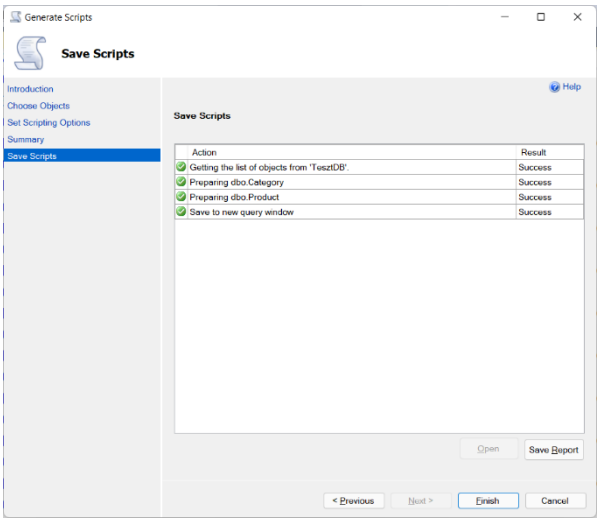
7. A következő képenyőn adhatjuk meg, hogy az export hova készüljün. Nekünk az ulsó opció lesz most jó. Egyébként az alábbi lehetőségek közül választhatunk
    - Save as notebook: Az eredményt egy ipynb fájlba menti le, amit aztán VS Code-dal vagy akár Google Colaboratory-val is meg lehet nyitni.
    - Save as script file: Itt megadhatjuk, hogy egy fájlba szeretnénk mindent, vagy objektum típusonként különbözőbe.
    - Save to clipboard: Az generálás eredménye a vágólapra kerül.
    - Open in new query window: Ilyenkor a management studioban nyit egy új oldalt és oda kerül a generált kód. Onnan el tudjuk menteni, vagy átmásolni bárhova.
8. Ezen felül még az Advanced gombra kell kattintani, ahol számos exportálási beállítást megadhatunk. Nekünk most a legfontosabb a **Type of data to script** beállítás, aho az alapértelmezet Schema only helyett válasszuk a **Data only** opciót.



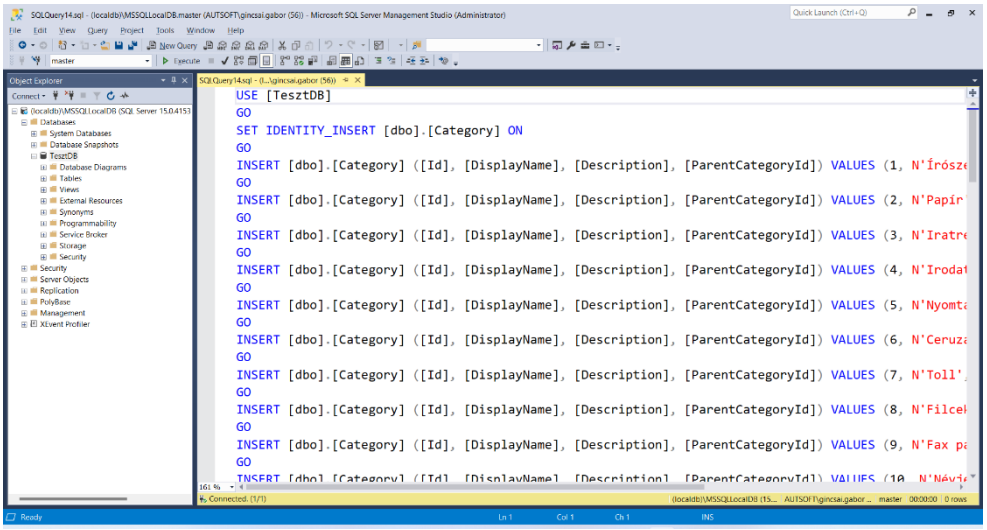


9. Ezt követően már csak összegzi a varázsló, hogy mit fog tenni és el is kezdi a munkát.

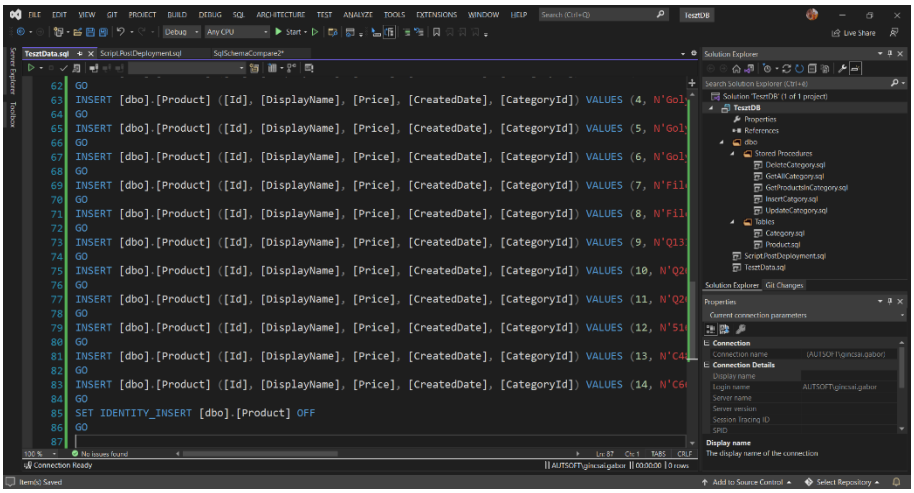




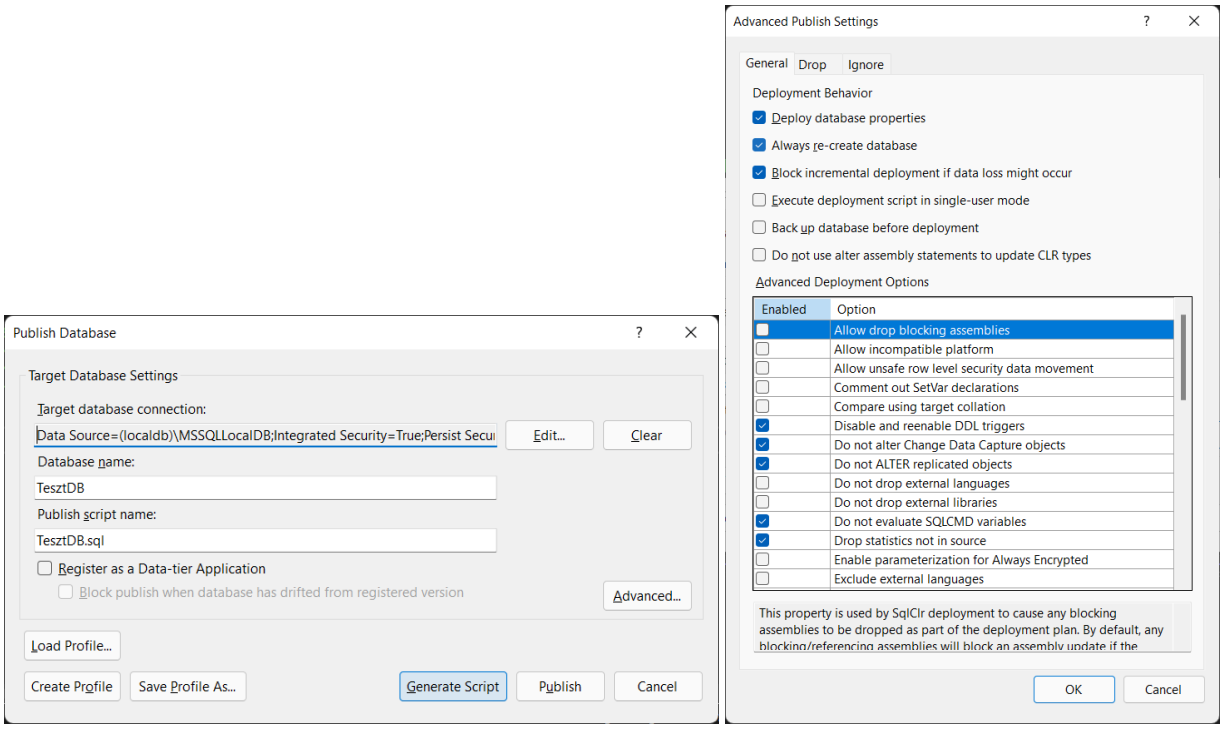
10. Ha végzett a generálással az alábbi képernyőn látható módon kapjuk meg az eredményt.



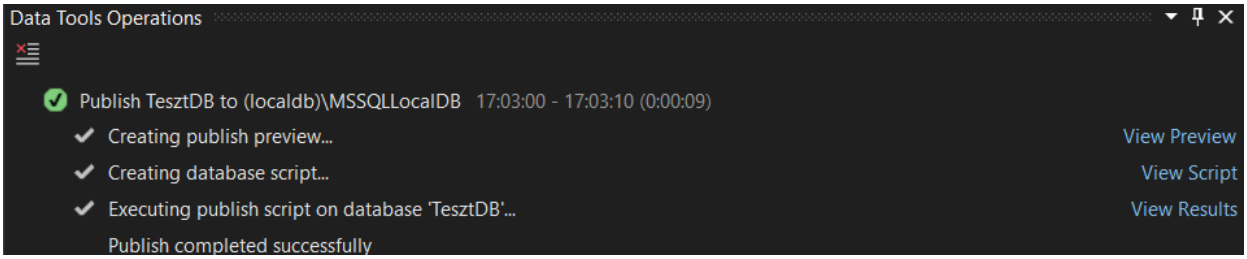
11. Ezt követően váltsunk vissza a TesztData.sql-re a Visual Studiban, majd másoljuk át a generált szkriptet.



12. Ezt követően publikáljuk az adatbázist. TestDB projekten jobb klikk >> *Publish*. Adjuk meg, hogy milyen connection striggel szeretnénk publikálni az adatbázist és az *Advanced* fül alatt állítsuk be, hogy mindig hozza létre újra az adatbázist.

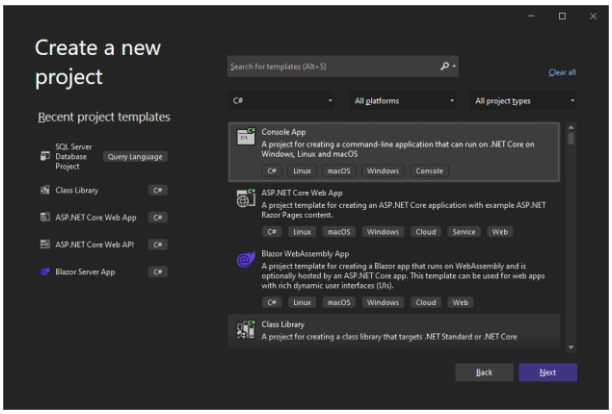


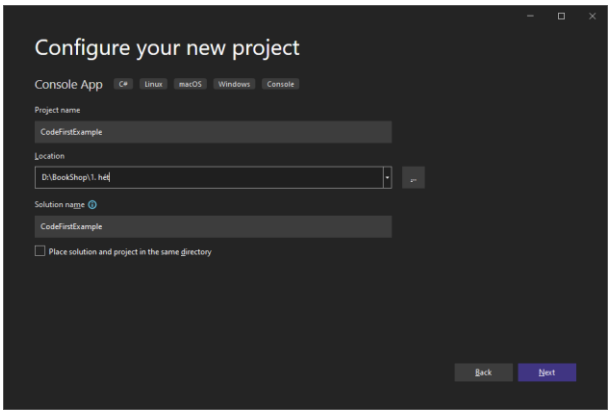
13. A *Save Profile As...* segítségével el tudjuk mentni a publikálási beállításokat például TestDB.publish névvel. Ekkor a solution-be bekerül egy TestDB.publish.xml így ha ismét telepíteni kell, akkor már elég erre duplán kattintani és minden beállítás meglesz a publikáláshoz. (Természetesen több publikálási beállítás is lehet egy adatbázishoz.)
14. Ez után már csak a *Publish* gombra kell kattintani, és meggyőződni, hogy minden sikerült. Ha valami hiba van, akkor a *View Result*-ra kattintva lehet kideríteni a pontos hibát.

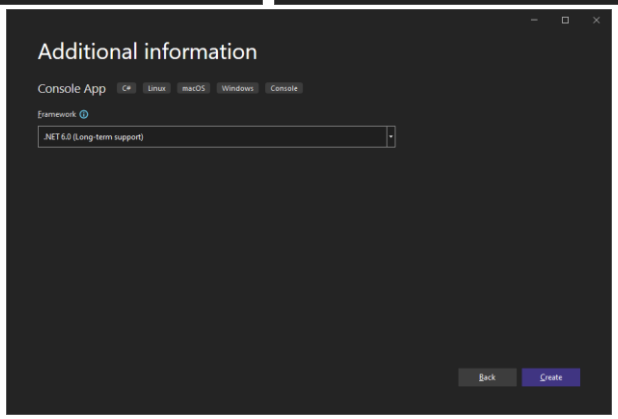


ADATBÁZIS KÉSZÍTÉSE EF CODE FIRST SEGÍTSÉGÉVEL

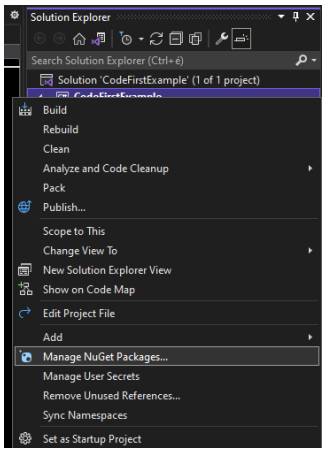
1. Hozzunk létre egy új konzolos alkalmazást, CodeFirstExemple névvel, C# nyelven.

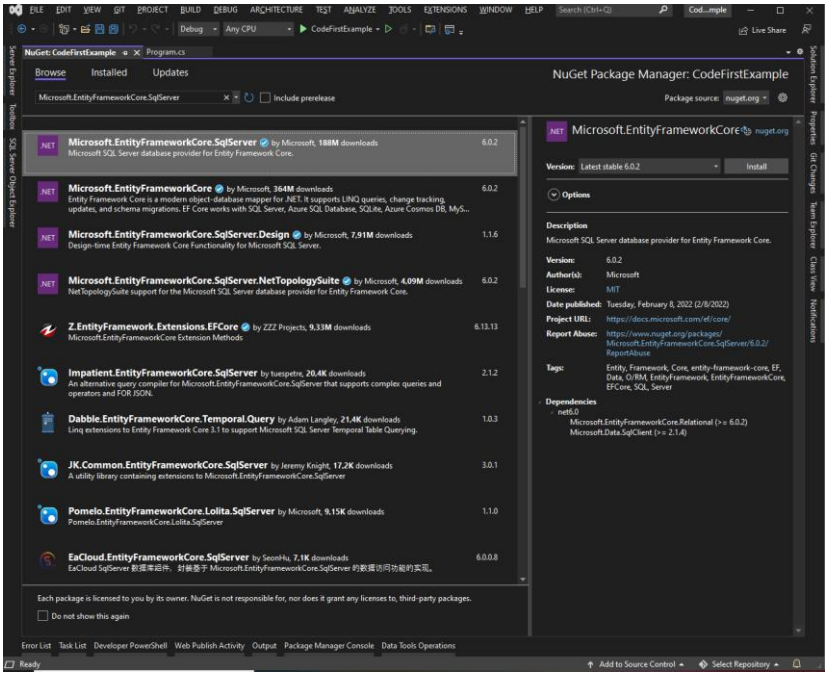






2. A Program.cs fájl már a top level statementeket használja, így csak egy sorból áll. Nincs benne Main függvény és semmi extra sem.
3. Adjuk hozzá a Microsoft.EntityFrameworkCore.SqlServer nuget packaget.





ADATBÁZIS MODELL LÉTREHOZÁSA

1. Hozzunk létre egy Entities mappát. Ebben fogjuk tenni majd az adatbázis entitásokat, azaz azokat az osztályokat, amiből majd az adatbázis táblákat generáljuk.
2. Adjuk hozzá a Category osztályt az alábbiak szerint.

```
using System.ComponentModel.DataAnnotations.Schema;

namespace CodeFirstExample.Entities
{
   public class Category
   {
      public Category()
      {
         Products = new HashSet<Product>();
         ChildCategories = new HashSet<Category>();
      }

      // Elnevezési konvenció alapján ha ID vagy classNameID, akkor Primary Key lesz.
      // Ha meg szeretnénk adni, hogy az ID-t ne a DB generálja.
      //[DatabaseGenerated(DatabaseGeneratedOption.None)]
      public int Id { get; set; }

      public string DisplayName { get; set; } = null!;
      public string? Description { get; set; }

      [ForeignKey("ParentCategory")]
      public int? ParentCategoryId { get; set; }
      public Category? ParentCategory { get; set; }
      public ICollection<Category> ChildCategories { get; set; }

      public virtual ICollection<Product> Products { get; set; }
   }
}
```

Mivel a .NET 6-os kódokban automatikusan be van kapcsolva a nullable reference type használata, ezért ha egy string típusú mezőt hozunk létre, akkor az alapértelmezés szerint nem lehet null. Azonban mivel minden referencia típus null-ra kerül inicializálásra ezért jelezni kell a kód elemzőnek, hogy tudjuk, hogy most ez null. Ezt a null forgiving opertatorral (!) lehet megtenni. Tehát a DiplayName esetén látható, hogy a get / set mögött szerepel egy = null! rész, amivel ki lehet küszöbölni a kódelemező warningját.
Ha azt szeretnénk, hogy a Description lehessen null, akkor string? -et kell megadni típusnak.
Figyeljük meg a kódban, hogy az alap tulajdonságok mellett megadtunk úgynevezett Navigation Property-ket. Ezek abban segítenek, hogy egy lekérdezésnél egyszerűen tudjunk hivatkozni a Category entitásból a benne lévő Productokra, vagy a szülő kategóriára.
A fenti kódban azt is láthatjuk, hogy a Category önmagára hivatkozik, mint ParentCategory. Ez adatbázis szinten megjelenik a ParentCategoryId mezőben. Azonban felvettük ennek a navigation propertynek a párját is, a ChildCategories-t, ami már egy lista. Így oda vissza irányba tudunk navigálni a kategória hierarchiában. Általános tanács, hogy a navigációs tulajdonságokat mindkét irányba vegyük fel, mert ez később a lekérdezések megírását nagy mértékben könnyíti.

3. Adjuk hozzá a Product modeljét.

```
namespace CodeFirstExample.Entities
{
   public class Product
   {
      public int Id { get; set; }

      [MaxLength(100)]
      public string DisplayName { get; set; } = null!;

      public int Price { get; set; }
      public DateTimeOffset CreatedDate { get; set; }
      public int CategoryId { get; set; }
      public Category Category { get; set; } = null!;
   }
}
```

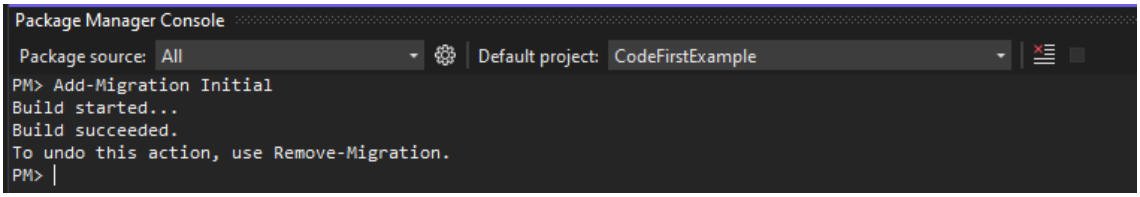
4. Hozzuk létre a projekt gyökér könyvtárába a kontextust inicializáló osztályt ShopDbContext névvel.

```
namespace CodeFirstExample
{
   public partial class ShopDbContext : DbContext
   {
      public ShopDbContext()
      { }
      public ShopDbContext(DbContextOptions options) : base(options)
      { }

      public DbSet<Product> Products => Set<Product>();
      public DbSet<Category> Categories => Set<Category>();

   protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
   {
      if( !optionsBuilder.IsConfigured)
         optionsBuilder.UseSqlServer(
         "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=ShopDb;Integrated Security=True");

      base.OnConfiguring(optionsBuilder);
   }

      protected override void OnModelCreating(ModelBuilder modelBuilder)
      {
         modelBuilder.Entity<Product>(entity =>
         {
            entity.Property(e => e.DisplayName).HasMaxLength(100);
         });

         base.OnModelCreating(modelBuilder);
      }
   }
}
```

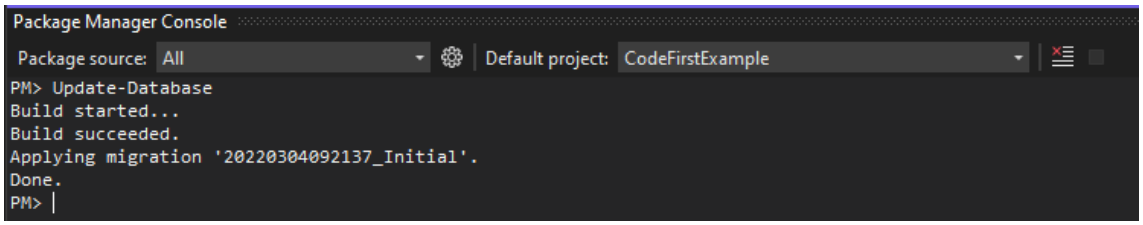
- A DbContext osztályból kell származni és érdemes az osztályt partial-ként létrehozni, hogy később akár egy külön fájlba szervezhessünk ki pl a teszt adatok feltöltését.

- Ebben az osztályban hozzuk létre a DbSet-eket, amik az adatbázis táblák lesznek. A táblák mezőit a DbSet típusa határozza meg. Amire itt is érdemes figyelni, hogy mivel ezek a DbSet-ek nem lehetnek null-ot ezért a => Set<Prodct>() rész odakerült a get; set; helyett.
- Az OnModelCreating metódust szoktuk felüldefiniálni, ahol a mezők hosszát, külső kulcsokat, vagy éppen, hogy a külső kulcsot mentén hogyan legyen a törlés szoktuk megadni. Jelen esetben mivel nem szeretnénk semmi extrát, csak annyi szerepel benne, hogy a Product-ban a DisplayName maximum 100 karakter lehet. Ha megnézzük a Category-ban ugyanezt a beállítást annotációval tettük meg. Érdemes arra figyelni, hogy vagy a fluent vagy az attribútum alapú megoldást válasszuk, de nem érdemes keverni a kettőt.
- Az OnConfiguration-t azért definiáltuk felül, mert ha nincs megadva connection string, akkor itt beállítjuk. ASP.NET-es alkalmazások esetén ezt nem használjuk, mert konfigurációból olvassuk fel az adatbázis kapcsolódási sztringet, de most az egyszerűség kedvéért itt adjuk meg. viszont ahhoz, hogy ez működjön a DbContext-ünknek kell, hogy legyen paraméter nélküli konstruktora is.

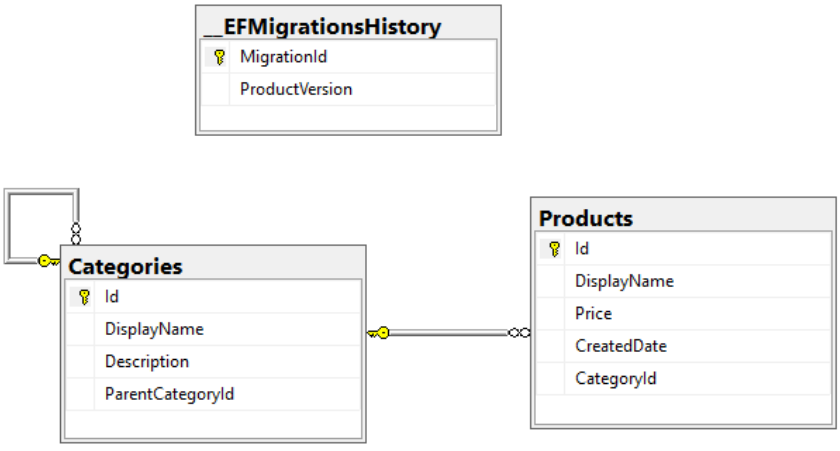
5. A kóddal el is készültünk, azonban ahhoz, hogy adatbázist tudjunk generálni a kódból készíteni kell egy migrációt.
   - Ahhoz, hogy ezt meg tudjuk tenni adjuk a projekthez a *Microsoft.EntityFrameworkCore.Tools* nuget packaget.
   - Ezt követően a Package Manager ablakból adjuk ki az Add-Migration Initial parancsot.



6. A migráció elkészítése után automatikusan megnyitja a Visual Studio a fájlt, amiben egy Up() és egy Down() függvényt találunk. Az Up() akkor fut le, ha a migráció korábban még nem futott le az adatbázison, a Down() pedig akkor ha egy korábbi verzióra kell visszaállni. Figyeljük meg, hogy az Up()-ban elkészültnek az adatbázis táblák és az elsődleges kulcsokhoz az indexek. Ezzel szemben a Down() csak eldobja a két táblát.
7. Ezt követően futtassuk is le az elkészített migrációt az Update-Database paranccsal a Package Manager Consoleban.



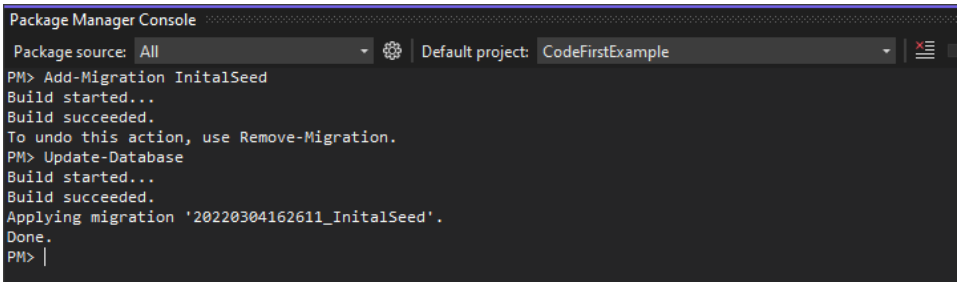
8. Ellenőrizzük, hogy a generált SQL adatbázisban minden kulcs a helyén van-e. Ehhez érdemes az adatbázisban egy diagramot létrehozni.



9. Adjunk hozzá az adatbázishoz teszt adatokat. Erre a legegyszerűbb módszer, ha a ShopDbContext-ben az OnModelCreating metódusnak a végén tesszük ezt meg. Természetesen ha sok tesztadat van, akkor azt érdemes lehet külön osztályba / metódusba kiszervezni, vagy akár úgy, hogy felvesszük a ShopDbContext partial osztály másik részét amiben csak a Seed van.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
	modelBuilder.Entity<Product>(entity =>
	{
		entity.Property(e => e.DisplayName).HasMaxLength(100);
		});

	// Seed data
	modelBuilder.Entity<Category>(entity => {
		entity.HasData(
			new Category { Id = 1, DisplayName = "Processzor" },
			new Category { Id = 2, DisplayName = "AMD", ParentCategoryId = 1 },
			new Category { Id = 3, DisplayName = "Intel", ParentCategoryId = 1 },
			new Category { Id = 4, DisplayName = "Billentyűzet" });
	});

	modelBuilder.Entity<Product>(entity => {
		entity.HasData(
			new Product { Id = 1, DisplayName = "X6 FX-6300", CategoryId = 2,
				Price = 25000, CreatedDate = DateTime.Now },
			new Product { Id = 2, DisplayName = "Core i7", CategoryId = 3,
				Price = 55000, CreatedDate = DateTime.Now },
			new Product { Id = 3, DisplayName = "Core i5", CategoryId = 3,
				Price = 35000, CreatedDate = DateTime.Now },
			new Product { Id = 4, DisplayName = "Core i3", CategoryId = 3,
				Price = 25000, CreatedDate = DateTime.Now });
	});

	base.OnModelCreating(modelBuilder);
}
```

10. Ahhoz, hogy ténylegesen be is szúrja ezeket az adatokat az adatbázisba készíteni kell egy migrációt, amit a Package Manager-ben kiadott **Add-Migtaion InitalSeed** segítségével tehetünk meg.
11. Ezt követően pedig egy **Updata-Database**-t kell még kiadni, hogy a DB-be is bekerüljenek az adatok.



TESZT LEKÉRDEZÉSEK

1. Mivel már vannak adatain így készíthetünk egy pár LINQ lekérdezést.
2. Adjunk a projekthez egy DbQueries mappát és abba egy TestQueries.cs fájlt, amibe a lekérdezéseket fogjuk írni.
3. Kérdezzük le az összes kategóriát. Mivel itt a DbContextet nem tudjuk DI-tól elkérni, mert arra nem készítettük fel a konzol alkalmazásunkat, így manuálisan példányosítjuk a ShopDbContextet, azonban ilyenkor fontos, hogy using blokkban használjuk, hogy a használat végén automatikusan Dispose-oljon.
   A lenti példában fontos megjegyezni, hogy maga az adatbázis lekérdezés akkor fut le, amikor a foreach ciklussal elkezdünk rajta végigiterálni.

```
public void ListAllCategories() {
      using var context = new ShopDbContext();
      var categories = from c in context.Categories select c;

      foreach (var c in categories)
            Console.WriteLine("ID: {0}\tName: {1}", c.Id, c.DisplayName);
}
```

4. Ahhoz, hogy ki is tudjuk próbálni a Program.cs-be készítsük el az alábbi kódot

```
using CodeFirstExample.DbQueries;

var categoryQuery = new TestQueries();
categoryQuery.ListAllCategories();
// TODO: További új függvények meghívása amit a többi lekérdezéshez készítünk
```

5. Ugyanezt a lekérdezést meg tudtuk volna írni függvényekkel is, mert a LINQ nem csak az SQL szerű szintaxist támogatja, hanem az alábbit is, amit nagyon gyakran használunk. Amit az SQL szintaxissal le lehet írni, azt az extension methodos formában is meg lehet adni.
   Itt fontos megjegyezni, hogy az adatbázis lekérdezés akkor fut le, amikor a ToList() meghívjuk.
   Mivel még egyelőre a teljes category entitást lekérjük a Select()-et fölösleges beleírni a kódba, hiszen nincs még szükség projekcióra.

```
public void ListAllCategories2() {
      using var context = new ShopDbContext();
      var categories = context.Categories.ToList();
      // var categories = context.Categories.Select(c => c).ToList();

      foreach (var c in categories)
            Console.WriteLine("ID: {0}\tName: {1}", c.Id, c.DisplayName);
}
```

6. Kérdezzük le egy kategóriát az ID-ja alapján.
   Itt pontosan egy elemet szeretnénk lekérdezni ezért a Single()-t használjuk.
   - **Single()**: Ha pontosan egy elem felel meg a szűrésnek visszaadja, egyébként hibát dob. Ha az adatbázisban a keresésnél megadott mezők együttesen egyediek, akkor ezt célszerű használni. (Pl ha elsődleges kulcs alapján keresünk)
   - **SingleOrDefault()**: Ha 0 vagy 1 elem fele meg a szűrésnek akkor visszadja, ha több akkor hibát dob.
   - **First()**: Visszaadja a szűrésnek megfelelő első elemet. Ha nincs megfelelő elem hibát dob.
   - **FirstOrDefault()**: Visszaadja a szűrésnek megfelelő első elemet. Ha nincs megfelelő elem akkor null-t ad.

```
public void GetProduct(int id)  {
      using var context = new ShopDbContext();
      var product = context.Products.Single(p => p.Id == id );

      Console.WriteLine("ID: {0}\tName: {1}", product.Id, product.DisplayName);
}
```

7. Kérdezzük le az adott kategóriában tartozó termékeket
   A kódban használunk az AsNoTracking() függvényt. Ez azt jelzi az Entity Frameworknek, hogy bár entitást ad vissza, de nem kell nyomonkövetnie a változásait, mert nem szeretnénk később a módosított tartalommal visszaírni az adatbázisba. Így gyorsabb a lekérdezés.

```
public void GetProductByCategory(int categoryId)
{
	using var context = new ShopDbContext();
	var products = context.Products
		.AsNoTracking()
		.Where(p => p.CategoryId == categoryId)
		.ToList();

	foreach (var p in products)
		Console.WriteLine("ID: {0}\tName: {1}", p.Id, p.DisplayName);
}
```

8. Módosítsuk úgy az előző lekérdezést, hogy aszinkron legyen és csak az ID és DisplayName mezőket kérje le az adatbázistól.
   A metódus async lett és a visszatérési értéke Task lett a void helyett. Erre azért van szükség, hogy a ToListAsync()-ot az await segítségével be tudjuk várni. A ToListAsync() lényege az, hogy aszinkron módon futtatja a DB lekérdezést.
   Ebben a kódban azért nem használjuk az AsNoTracking()-et mert van egy projekció tehát nem az entitást adjuk vissza, így nem kell jelezni külön az EF-nek, hogy ne kövesse az entitást, hiszen nem azt adunk vissza.
   Figyeljük meg, hogy a Select-ben egy transzformációt hajtunk végre, ahol a Category (c) lesz egy olyan anonim típus, aminek egy Id és DisplayName tulajdonsága van.

```
public async Task GetProductByCategoryAsync(int categoryId)
{
	using var context = new ShopDbContext();
	var products = await context.Products
		.Where(p => p.CategoryId == categoryId)
		.Select(p => new { p.Id, p.DisplayName })
		.ToListAsync();

	foreach (var p in products)
		Console.WriteLine("ID: {0}\tName: {1}", p.Id, p.DisplayName);
}
```

9. Hozzunk létre egy új terméket.
   Először meg kell példnyosítani az entitásunkat jelen esetben a Prduct-ot és beállítani az egyes tulajdonságok értékét. Arra kell figyelni, hogy az Id-t az adatbázis generálja, tehát az nem kell beállítani, illetve mi most a CategoryId-t állítjuk be egy létező értékre, tehát a Category-t nem kell beállítani.

```
public async Task AddProductAsync( string displayName, int price, int categoryId) {
	var newProduct = new Product
	{
		DisplayName = displayName ,
		Price = price,
		CategoryId = categoryId
	};

	using var context = new ShopDbContext();
	context.Products.Add(newProduct);
	await context.SaveChangesAsync();
}
```

10. Adjunk hozzá egy új terméket úgy, hogy csak a kategória nevét ismerjük.
    Ebben az esetben csak a kategória nevét ismerjük. Először le kell kérdezni a kategóriát. Itt mi az egyszerűség kedvéért Single()-t használunk, de ha előfordulhat, hogy nem létezik a kategória, akkor SingleOrDefault()-ot kellene használni és vizsgálni, hogy ha nincs ilyen kategória akkor mi történjen.
    Miután lekérdeztük a kategória entitást, példányosítjuk a terméket és beállítjuk a tulajdonságait. Ebben az esetben azonban nem a CategoryId-t állítjuk be, hanem a Category-t hiszen a teljes entitás megvan.
    Akkor lehet fontos, hogy az entitást állítsuk be és nem az Id-ját ha például azt is most szúrjuk be és még nem tudjuk mi lesz az Id-ja hiszen azt az adatbázis fogja generálni. Sokat ilyenkor fölöslegesen hívnak egy SaveChages()-t hogy meglegyen az Id holott nincs rá szükség ha megvan az entitás (akár Id nélkül is).

```
public async Task AddProductToCategoryAsync(string categoryName, string dispalyName, int price)
{
      using var context = new ShopDbContext();
      var category = context.Categories.Single(c => c.DisplayName == categoryName);

      var newProduct = new Product {
            DisplayName = dispalyName,
            Price = price,
            Category = category
      };

      context.Products.Add(newProduct);
      await context.SaveChangesAsync();
}
```

11. Módosítsunk egy terméket az Id-ja alapján
    A módosításhoz először le kell kérdezni az elemet a DB-ből és nem szabad az AsNoTrackinget() használni. Majd be kell állítani a módosítani kivált tulajdonságokat a lekérdezett entitások meg meghívni a SaveChagesAsync()-ot a változások elmentéséhez.

```
public async Task UpdateProductAsync(int id, string displayName, int price) {
      using var context = new ShopDbContext();
      var product = context.Products.Single(p => p.Id == id);

      product.DisplayName = displayName;
      product.Price = price;

      await context.SaveChangesAsync();
}
```

12. Töröljünk terméket az ID-ja alapján.
    A törölni kívánt terméket először lekérdezzük az adatbázisból. Single()-t használunk, mert ha nem létezik a termék, akkor nyugodtan dobhat kivételt. Ezt követően a Remove() segítségével eltávolítjuk az elemet, majd a SaveChagesAsnyc()-kal elmentjük a változást.

```
public async Task DeleteProductAsync(int id) {
      using var context = new ShopDbContext();
      var product = context.Products.Single(p => p.Id == id);
      context.Remove(product);

      await context.SaveChangesAsync();
}
```

13. Töröljük a terméket az ID-ja alapján úgy, hogy előtte nem kérdezzük le.
    Mivel ismerjük az elsődleges kulcsát a terméknek ezért nem kell külön lekérdezni a teljes entitást, mert a törlésnél úgyis csak az elődleges kulcsot nézni. Így meg tudunk spórolni egy adatbázis lekérdezést a törlésnél.

```
public async Task DeleteProductAsnyFast(int id) {
      using var context = new ShopDbContext();
      context.Remove(new Product { Id = id });

      await context.SaveChangesAsync();
}
```

14. A teljes Program.cs fájl az alábbiak szerint néz ki, ami minden függvényt meghív. Mivel ID-val hivatkozunk az elemekre érdemes belepillantani az adatbázisba, hogy helyes értékkel hívjuk meg, illetve a teszteléshez érdemes egyesével hívni, hogy átláthatóbb legyen az eredmény.

```
using CodeFirstExample.DbQueries;

var categoryQuery = new TestQueries();

categoryQuery.ListAllCategories();
categoryQuery.ListAllCategories2();

categoryQuery.GetProduct(3);
categoryQuery.GetProductByCategory(3);
await categoryQuery.GetProductByCategoryAsync(3);

await categoryQuery.AddProductAsync("Logitech billentyűzet", 12000, 4);
await categoryQuery.AddProductToCategoryAsync("Intel", "Core i9", 195000);

await categoryQuery.UpdateProductAsync(5, "Módosított", 12541);
await categoryQuery.DeleteProductAsync(5);
await categoryQuery.DeleteProductAsnyFast(6);
```