Optional 5b : Entity Framework Code First From Database

Objective

In the previous lab you did Code First – i.e. start with the code and construct the database from it. In this lab, we do it the other way around – start with the Database and build the code from the database schema.

This exercise will take around 40 minutes.

1 Start a new Core Console App called EFFromDatabase and add the following NuGet packages:
Microsoft.EntityFrameworkCore.SqlServer
Microsoft.EntityFrameworkCore.Tools
Bricelam.EntityFrameworkCore.Pluralizer

(without the last one, the scaffolder will produce plural-named classes eg Customer**S**)

2 Ensure you have the Northwind database installed in .\SqlExpress.
If not, there is a SQL script for installing is in the Assets folder

3 Open a command window at the project folder (as we've done in previous labs) and enter this (it must all be on 1 line so drop it into Notepad first and check no new lines and regular quotes – not "xx" type of quotes).

```
dotnet ef dbcontext scaffold
"Server=.\SqlExpress;Database=Northwind;Trusted_Connection=True"
Microsoft.EntityFrameworkCore.SqlServer -o Models --context NorthwindContext

May need:
dotnet ef dbcontext scaffold
"Server=.\SqlExpress;Encrypt=False;Database=Northwind;Trusted_Connection=True"
Microsoft.EntityFrameworkCore.SqlServer -o Models --context NorthwindContext
```

This should produce all classes into a folder named 'Models'

Note: if you get an error "Could not execute because the specified command or file was not found", then you will need to install the dotnet ef tool. You can do this by typing the following into the command line. After this succeeds, try the previous command again:

```
dotnet tool install --global dotnet-ef --version 3.1.3
```

If necessary, modify this command so that the version matches the version of Entity Framework that you installed.

4 Have a look at the Customer class and note:
It's a partial class
There are no fields – its all properties
Collections are virtual

5 Enter this into Main() and run it

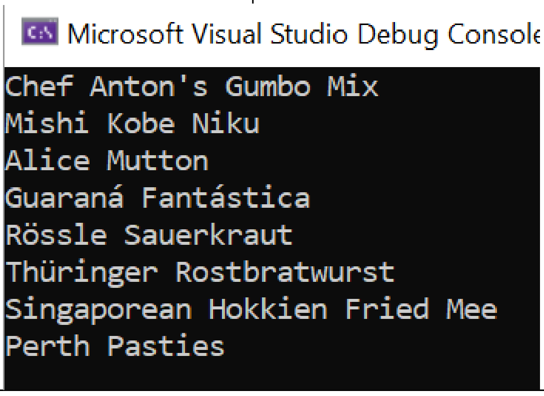
```
using (NorthwindContext ctx = new NorthwindContext())
{
    foreach (Customer c in ctx.Customers
                            .Include(c => c.Orders)
                            .ThenInclude(o => o.OrderDetails)
                            .ThenInclude(od => od.Product))
    {
        Console.WriteLine(c.ContactName);
        foreach (Order o in c.Orders)
        {
            Console.WriteLine("..." + o.OrderId);
            foreach (OrderDetail od in o.OrderDetails)
            {
                Console.WriteLine("......" + od.Product.ProductName);
            }
        }
    }
}
```
You should get lots of Order and Order Details

Query Filters

6 Comment-out all the above code in Main() and add in this code

```
using (NorthwindContext ctx = new NorthwindContext())
{
    var discontinued = ctx.Products.Where(p => p.Discontinued).ToList();
    discontinued.ForEach(d => Console.WriteLine(d.ProductName));
}
```

Run it – it will output a list of all discontinued products:



```
Chef Anton's Gumbo Mix
Mishi Kobe Niku
Alice Mutton
Guaraná Fantástica
Rössle Sauerkraut
Thüringer Rostbratwurst
Singaporean Hokkien Fried Mee
Perth Pasties
```

7 But suppose you *never* wanted to see discontinued products in any of your queries. Right now you'd have to put in a Where statement into every query. EFCore has a QueryFilter feature where you can do this globally.

Go to the Northwind context class and search for <Product> (include the angle brackets in the search)

When you reach this line

```
modelBuilder.Entity<Product>(entity =>
{
```

Somewhere in this section add a global query filter

```
entity.HasQueryFilter(e=>!e.Discontinued);
```

8 Run again and this time your query won't even see discontinued products.

9 If, in particular queries, you really did want to see the discontinued products, you can override this (in Program/Main()) :

```
ctx.Products.Where(p => p.Discontinued).IgnoreQueryFilters().ToList();
```

Do this and confirm you again see the discontinued products.

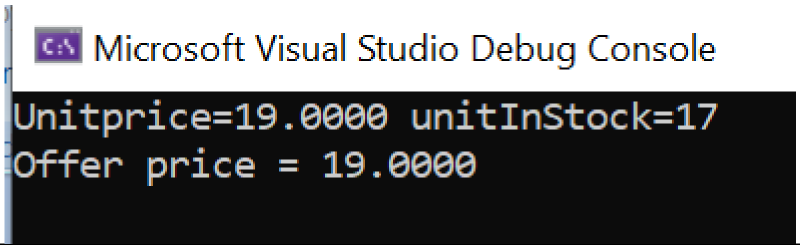
10 Remove the Query Filter you just added to NorthwindContext and comment-out all code in Main()

Adding own functions to database classes

11 Suppose that the company wanted to sell off discontinued lines at half price. Comment out existing code in Main() and add this:

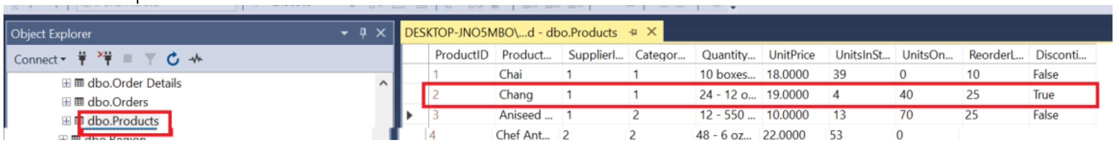
```
using (NorthwindContext ctx = new NorthwindContext())
{
    Product chang = ctx.Products
                          .Where(p => p.ProductName == "Chang").Single();
    Console.WriteLine(
       $"Unitprice={chang.UnitPrice } unitInStock={chang.UnitsInStock}");

    decimal? offerPrice = chang.UnitPrice;
    if (chang.UnitsInStock < 5 && chang.Discontinued)
    {
        offerPrice /= 2;
    }
    Console.WriteLine($"Offer price = {offerPrice}");
}
```

Run this
The offer price is 19 because this produce does not meet the UnitsInStock<5 and Discontinued conditions.



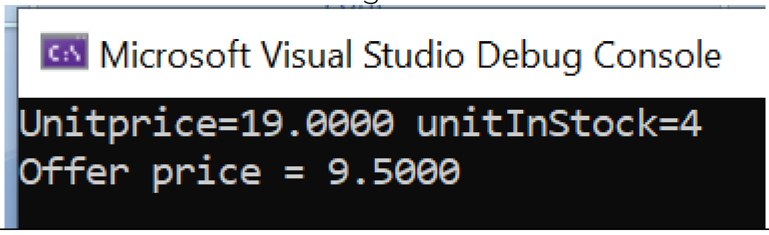
```
Microsoft Visual Studio Debug Console
Unitprice=19.0000 unitInStock=17
Offer price = 19.0000
```

12 Open up the Products table in SSMS by right-clicking and “Edit top 200 rows”. Find the product



ProductID	Product...	SupplierI...	Categor...	Quantity...	UnitPrice	UnitsInSt...	UnitsOn...	ReorderL...	Disconti...
1	Chai	1	1	10 boxes...	18.0000	39	0	10	False
2	Chang	1	1	24 - 12 o...	19.0000	4	40	25	True
3	Aniseed ...	1	2	12 - 550 ...	10.0000	13	70	25	False
4	Chef Ant...	2	2	48 - 6 oz...	22.0000	53	0		

Change the Units in stock to 4 and set the Discontinued status to True to show the above code is working



```
Microsoft Visual Studio Debug Console
Unitprice=19.0000 unitInStock=4
Offer price = 9.5000
```

13 It would be good to move this code into the Product class – that’s where it really belongs.
Open the Product class and add

```
public decimal? OfferPrice =>
   (UnitsInStock < 5 && Discontinued) ? UnitPrice /= 2 : UnitPrice;
```

Have this as your code in Main()

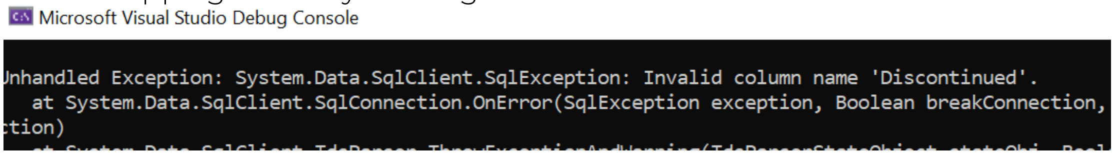
```
using (NorthwindContext ctx = new NorthwindContext())
{
    Product chang = ctx.Products
                           .Where(p => p.ProductName == "Chang").Single();
    Console.WriteLine(
        $"Unitprice={chang.UnitPrice } unitInStock={chang.UnitsInStock}");
```

	```
	    Console.WriteLine($"Offer price = {chang.OfferPrice}");
	}
	```
	Confirm it still works.

14	Now imagine that we want to delete the Discontinued column completely from the database. If we did this, we would need to re-generate our classes (imagine there might be other subtle changes we didn't necessarily know about). Doing this would lose the OfferPrice code we've just added to the Product class as this is going to get blown away.
15	To solve the first one, add a folder Logic alongside Models. Hold down the Ctrl key and drag Product into Logic (ie copy it)
16	Delete OfferPrice from Models/Product and delete everything except OfferPrice from Logic/Product

17	Suppose we had an additional constraint that ProductName must be shorter than 50 characters. We can accommodate this too.
	Make your Logic/Product file look like this:
	`using Microsoft.AspNetCore.Mvc;`
	`using System;`
	`using System.Collections.Generic;`
	`using System.ComponentModel.DataAnnotations;`
	
	`namespace EFFromDatabase.Models`
	`{`
	`    [ModelMetadataType(typeof(Product_Buddy))]`
	`    public partial class Product`
	`    {`
	`        public decimal? OfferPrice =>`
	`          (UnitsInStock < 5 && Discontinued) ? UnitPrice /= 2 : UnitPrice;`
	`    }`
	
	`    public class Product_Buddy`
	`    {`
	`        [MaxLength(50)]`
	`        public string ProductName { get; set; }`
	`    }`
	`}`
	If you Ctrl+dot ModelMetadataType it will suggest you install Microsoft.AspNetCore.Mvc.Core. Install this then resolve namespaces.
	Ie this extra constraint is available to the MVC validation system.
	Run and make sure all is still working

18	Now, in SSMS, right-click the Products table > Design and delete the Discontinued column. Save the table

19	Run the app again and you will get
	
	Microsoft Visual Studio Debug Console
	`Jnhandled Exception: System.Data.SqlClient.SqlException: Invalid column name 'Discontinued'.`
	`   at System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection,`
	`tion)`
	Imagine it wasn't just the one change and that now we need to do a complete re-scaffolding

20	Open a command window at the project directory and enter `dotnet ef dbcontext scaffold "Server=.\SqlExpress;Database=Northwind;Trusted_Connection=True" Microsoft.EntityFrameworkCore.SqlServer -o Models --context NorthwindContext –force` Check that the 'Discontinued' property has gone from the Product class

21	If you now compile, you can fix the 2 ‘Discontinued’ issues, Run and you’re back in business! `public decimal? OfferPrice => (UnitsInStock < 5) ? UnitPrice /= 2 : UnitPrice;`