# Working with data part 1

This lesson focuses on working with SQL-Server using [Entity Framework Core](https://docs.microsoft.com/en-us/ef/core/).

Entity Framework (EF) Core is a lightweight, extensible, open source and cross-platform version of the popular Entity Framework data access technology.

EF Core can serve as an object-relational mapper (O/RM), which:

* Enables .NET developers to work with a database using .NET objects.
* Eliminates the need for most of the data-access code that typically needs to be written.

EF Core began life as Entity Framework (minus Core) which reached end of life at version 6. Entity Framework 6 is used in millions of applications where developers in many cases find upgrading to EF Core to much work as the architecture is very different between EF and EF Core for the better.

The default method to work with data in ASP.NET Core solution is EF Core.

**Notes**

* EF Core first hit takes longer than a data provider but after the first hit is equal or better performance.
* Data provider access requires a good deal more hand coding

https://github.com/dotnet/EntityFramework.Docs

# DbContext

The [DbContext](https://docs.microsoft.com/en-us/ef/core/dbcontext-configuration/) class is simply the way for the developers to incorporate Entity Framework based data to the application.

* It allows you to make database connections inside an application model and allows the developer to link the model properties to the database table using a connection string.
* It is the base class to manage all types of database operations, such as establishing a connection with the database, query the database and end the connection.

# Setup SSMS

With [NorthWind database](https://gist.github.com/karenpayneoregon/40a6e1158ff29819286a39b7f1ed1ae8). Karen will step everyone through this.

# Preparing data operations using Entity Framework Core

1. From Visual Studio menu, File -> New -> Project
2. Select Blank Solution, select Next.
3. Solution name: NorthWindCoreSolution
4. Location: C:\OED\Dotnetland\VS2019
5. Click Create button

Make sure Solution Explorer window is open

From Visual Studio menu, View -> Solution Explorer. Once displayed, pin to the side (Karen will talk through this)

1. Right click on the top node in Solution Explorer, select Add -> New Solution folder
2. Name the folder Class projects. (Solution folders are virtual, not physical)
3. Right click on the Class projects node and select Add -> New Project.
   1. Select BaseNetCoreProject. BaseNetCoreProject is a custom project template which hopefully is installed by the service desk and if not we will work through this.
4. Click the Next button
5. Project name: NorthWindCoreLibrary
6. Click the Create button.
7. The project appears with several folders.
8. Click on the Classes folder, delete class1.cs
9. Right click in solution explorer on the project name.
10. Select EF Power Tools -> Reverse engineer
    1. Click the Add button
    2. Server Name: .\SQLEXPRESS
    3. Click the drop down for Select or enter a database name.
    4. Select NorthWind2020
    5. Click Test Connection button, click OK (hopefully this works w/o permission issues)
    6. Click the checkbox for Use EF Core 5.
    7. Click OK button
    8. A selection object dialog appears, click the checkbox in the upper left corner. We don’t need all objects but will keep it simple.
    9. Click the OK button
    10. Context name: NorthwindContext
    11. Entity Type path: Models
    12. DbContext Path: Data
    13. Include connection string: check
    14. Click Advance button
    15. Split DbContext into Configuration classes: Check
    16. Click OK button then click OK again
11. Talk through Getting ConfigurationHelper project into the solution
12. Talk through appsetting.json in NorthWindCoreLibrary
13. Select the Dependencies folder in NorthWindCoreLibrary project
14. Right click, Add project reference
15. Select ConfigurationHelper, click OK.
16. Select NorthWindContext.cs under the Data Folder.
17. Add the following as the top line “using ConfigurationHelper;”
18. Find the method OnConfiguring
19. Change optionsBuilder.UseSqlServer to optionsBuilder.UseSqlServer(Helper.ConnectionString());
20. Delete the line starting with #warning
21. Right click the top node in Solution Explorer, select Rebuild Solution.

# Setting up Unit Test

1. Right click on solution explorer select Add -> New Solution Folder
2. Enter Unit Test projects
3. Right click on the new solution folder
4. Select Add -> New project -> BaseCoreUnitTestProject, click Next button
5. Name the project: NorthWindCoreUnitTest, click the Create button
6. Right click the Dependencies folder, add project reference
7. Select: NorthWindCoreLibrary, click OK button
8. Double click the NorthWindCoreLibrary project which open the configuration
9. Remove –windows, save the file.
10. Right click NorthWindCoreLibrary, select Rebuild.
11. Double click MainTest.cs in NorthWindCoreUnitTest project.
12. Replace the contents with the following (we will discuss the code before running the code)

using System.Diagnostics;

using System.Linq;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using NorthWindCoreLibrary.Data;

using NorthWindCoreUnitTest.Base;

namespace NorthWindCoreUnitTest

{

    [TestClass]

    public partial class MainTest : TestBase

    {

        [TestMethod]

        [TestTraits(Trait.PlaceHolder)]

        public void TestMethod1()

        {

            using var context = new NorthwindContext();

            var customers = context.Customers.ToList();

            Debug.WriteLine(customers.Count);

            Assert.IsTrue(customers.Count == 91);

        }

    }

}

* At this point there may be differences between editions of Visual Studio Pro vs Enterprise editions

1. From Visual Studio menu, select Test -> Test Explorer
2. Pin the Test Explorer window (Karen will walk through this)
3. In Test Explorer upper left corner, select the first green arrow which will run the test. Once done there should be a green check mark next to TestMethod1
4. Next, let’s rename TestMethod1 to CustomerCount
5. Test Explorer will show the name change.
6. Change [TestTraits(Trait.PlaceHolder)] to [TestTraits(Trait.CustomersSelect)] (Karen will adhoc this as she has an extension that handles this)
7. Since the test name has change there is no green arrow, run the test again.
   1. Did the test run? Was there an issue with TestMethod1 does not exists?
8. Now rename the test class to CustomersTest
9. Run the test again

# Refactoring configuration classes

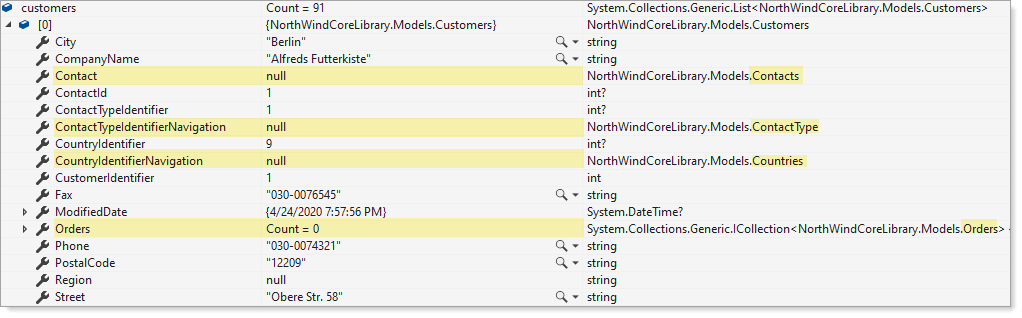
Under the project NorthWindCoreLibrary, folder Data there are classes for each table in the SQL-Server database, let’s move them to clear things up. Note that if the reverse engineering runs again we need to reverse this next process.

1. Right click on the Data Folder -> Add new Folder, name the folder Configurations
2. Select each class in the data folder except NorthwindContext.cs and drag them into the Configuration folder.
3. If we are sure the models will not change then we can alter the namespaces for all models in the configurations folder (Karen will demonstrate) but let’s assume they will.

* Walkthrough the code in NorthWindCoreLibrary
* Walkthrough the code in NorthWindCoreUnitTest
* Explain ConfigurationHelper code and how this should be part of a team library or a local NuGet package.
* Add a solution readme.md file, explain what goes in it.

# Back to our unit test

* Walkthrough setting a breakpoint on Debug.WriteLine and examine customer data in the local window



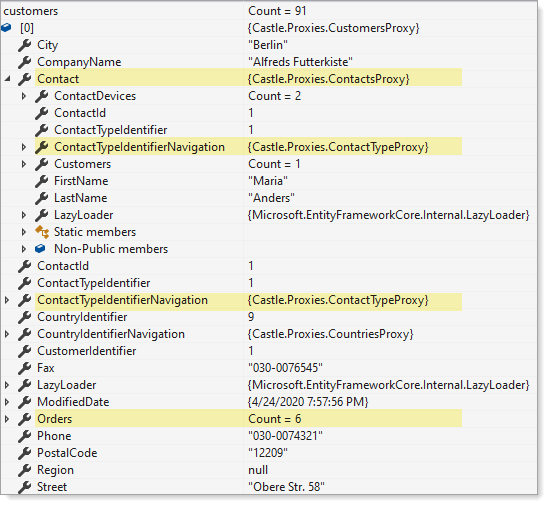
Note that highlighted properties are null. This is the default behavior to only load what has been asked for. There are several options.

The first option is to load all navigation properties e.g.

* Contact
* ContactTypeIdentifierNavigation
* CountryIdentifierNavigation
* Orders

This is done by first adding a NuGet package known as [Microsoft.EntityFrameworkCore.Proxies](https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Proxies/) using *NuGet Package Manager* or *Package Manager Console*.

* Open NorthWindCoreLibrary.Data. NorthwindContext class.
* Change optionsBuilder.UseSqlServer(Helper.ConnectionString()); to optionsBuilder.UseLazyLoadingProxies().UseSqlServer(Helper.ConnectionString());
* Rebuild the Visual Studio solution
* Rerun the test method CustomerCount using Debug. We now have all navigation properties populated



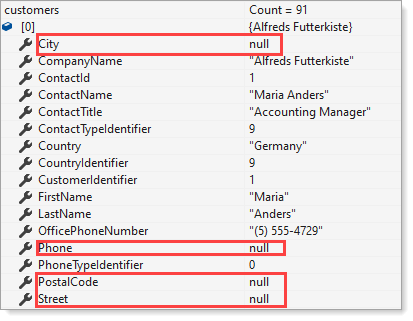
Downside, if all the navigations are not needed our payload is heavier than needed.

* Change back to optionsBuilder.UseSqlServer(Helper.ConnectionString());.
* In NorthWindCoreLibrary, create a new folder named Projections.
* Add a class named CustomerItem (Karen will provide the code)
* Create a new class in the Classes folder named CustomersOperations.
* Karen will provide code for a method named GetCustomersWithProjectionAsync and walkthrough the code.
* Back in NorthWindCoreUnitTest. CustomersTest add a test method, type in the follow (or Karen can give it to you)

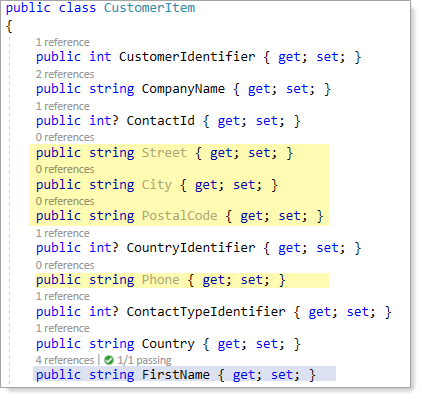
|  |
| --- |
| [TestMethod]  [TestTraits(Trait.EfCoreCustomersSelect)]  public async Task CustomersProject()  {      var customers =          await CustomersOperations.GetCustomersWithProjectionAsync();        string firstName = customers          .FirstOrDefault(cust => cust.FirstName == "Maria").FirstName;        Assert.IsTrue(firstName == "Maria");  } |

* Discussion on
  + Projections (refer to image below)
  + async
  + await

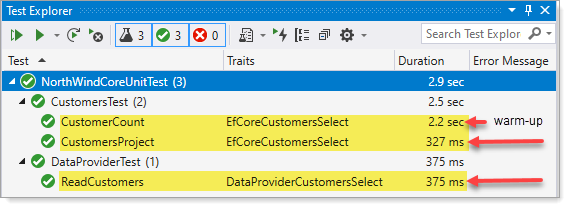
**Returned data**



Visually seeing what properties have been used and what has been tested



Execution time, note EF Core test CustomerCount is 2.2 seconds, this is the first call to EF Core which is a warm-up. Then note the time on the second test CustomersProject, same time as with ReadCustomers (using a data provider), they are both equal in execution time.



# GitHub

Walkthrough creating a new repository and create/push solution