



FREE eBook

LEARNING

Entity Framework Core

Free unaffiliated eBook created from
Stack Overflow contributors.

**#entity-
framework-
core**

Table of Contents

About.....	1
Chapter 1: Getting started with Entity Framework Core.....	2
Remarks.....	2
Examples.....	2
Adding packages to the project.....	2
Database First in Entity Framework Core with a Class Library and SQL Server.....	2
Step 1 - Install .NET Core.....	2
Step 2 - Create The Projects.....	3
Step 3 - Installing EF Packages.....	5
----- OR.....	6
Step 4 - Creating the Database Model.....	7
Finally.....	9
Passing a Connection String.....	10
Model, Querying and Saving Data.....	11
Model.....	11
Querying.....	11
Saving Data.....	12
Deleting Data.....	12
Updating Data.....	12
Chapter 2: EF Core vs EF6.x.....	14
Remarks.....	14
Examples.....	14
Side-by-side comparison.....	14
Chapter 3: Updating a Many to Many relationship.....	18
Introduction.....	18
Examples.....	18
MVC POST Edit example.....	18
Credits.....	20

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [entity-framework-core](#)

It is an unofficial and free Entity Framework Core ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Entity Framework Core.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Entity Framework Core

Remarks

Entity Framework (EF) Core is a lightweight and extensible version of the popular Entity Framework data access technology.

EF Core is an object-relational mapper (O/RM) that enables .NET developers to work with a database using .NET objects. It eliminates the need for most of the data-access code that developers usually need to write.

Examples

Adding packages to the project

To add EntityFrameworkCore to your project, update the `project.json` file (add new lines into the `dependencies` and `tools` sections):

```
"dependencies": {  
  ...  
  "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",  
  "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",  
  "Microsoft.EntityFrameworkCore.Design": {  
    "version": "1.0.0",  
    "type": "build"  
  },  
},  
"tools": {  
  ...  
  "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"  
}
```

Don't forget to run `dotnet restore` to actually download these packages from the internet.

If you are using an RDBMS other than Microsoft SQLServer - replace

`Microsoft.EntityFrameworkCore.SqlServer` with the correct version (

`Microsoft.EntityFrameworkCore.Sqlite`, `Npgsql.EntityFrameworkCore.PostgreSQL` or other - consult your RDBMS documentation for the recommended package).

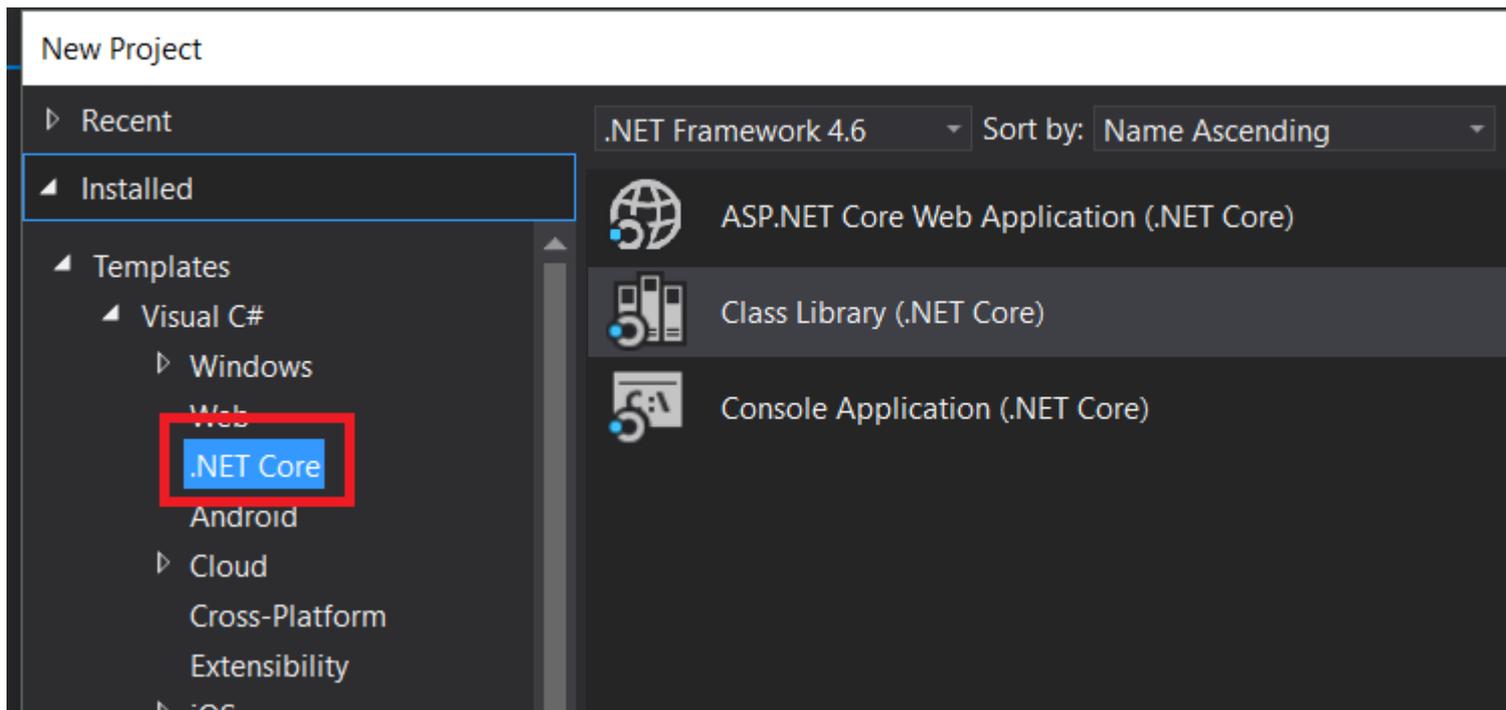
Database First in Entity Framework Core with a Class Library and SQL Server

Okay it took me about a day to figure it out so here I am posting the steps I followed to get my Database First working in a `Class Project (.NET Core)`, with a `.NET Core Web App`.

Step 1 - Install .NET Core

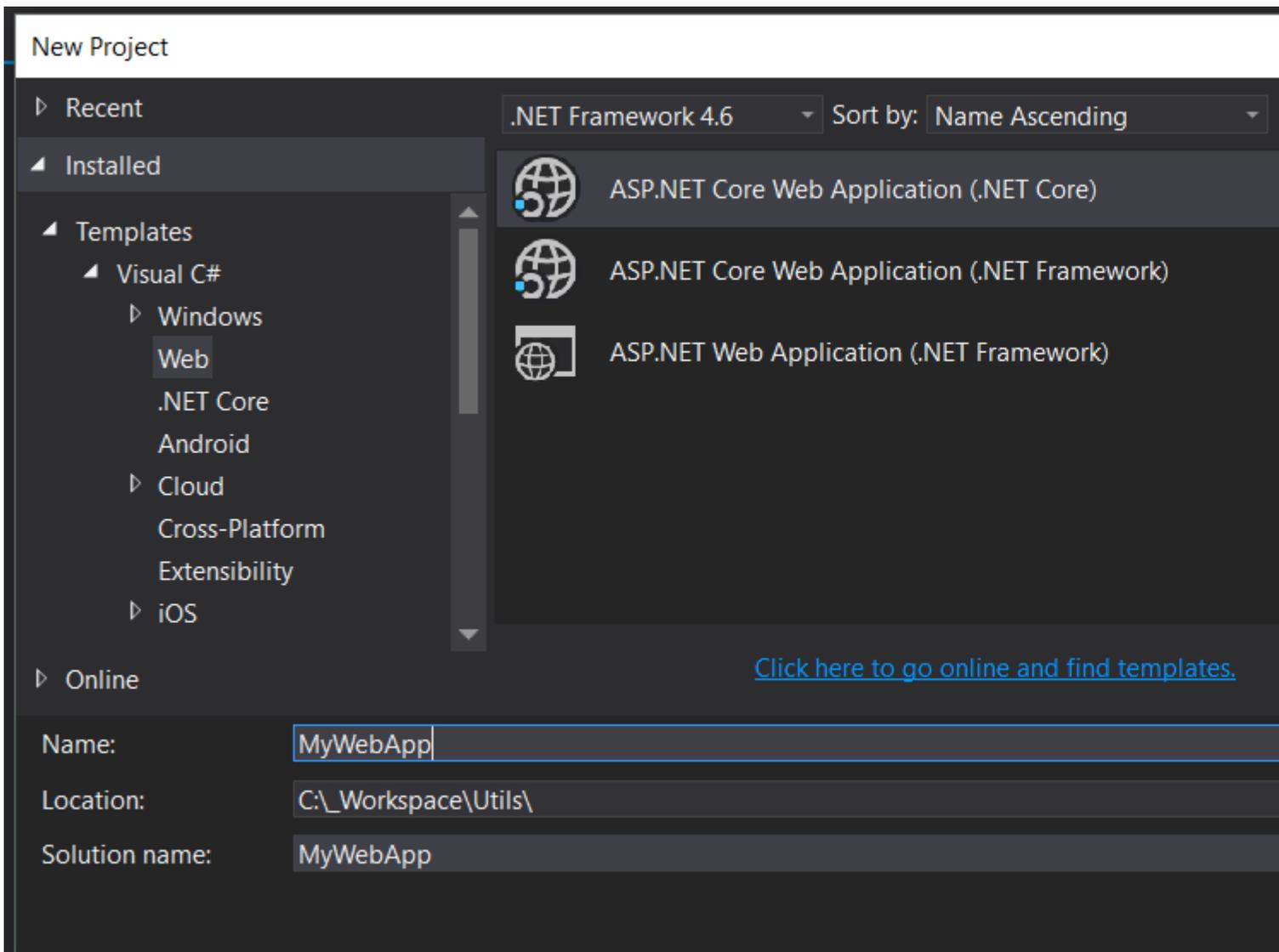
Make Sure you are using .NET Core not DNX (Hint: You should be able to see the .NET Core option when creating a New Project) - If NOT Download from [Here](#)

If you have problems installing .NET Core (Error is something like Visual Studio 2015 Update 3 not installed correctly) - You can run the installing using the command: [DotNetCore.1.0.0-VS2015Tools.Preview2.exe SKIP_VSU_CHECK=1] -- Which will prevent the installation performing the Visual Studio Check [Github Issue](#)

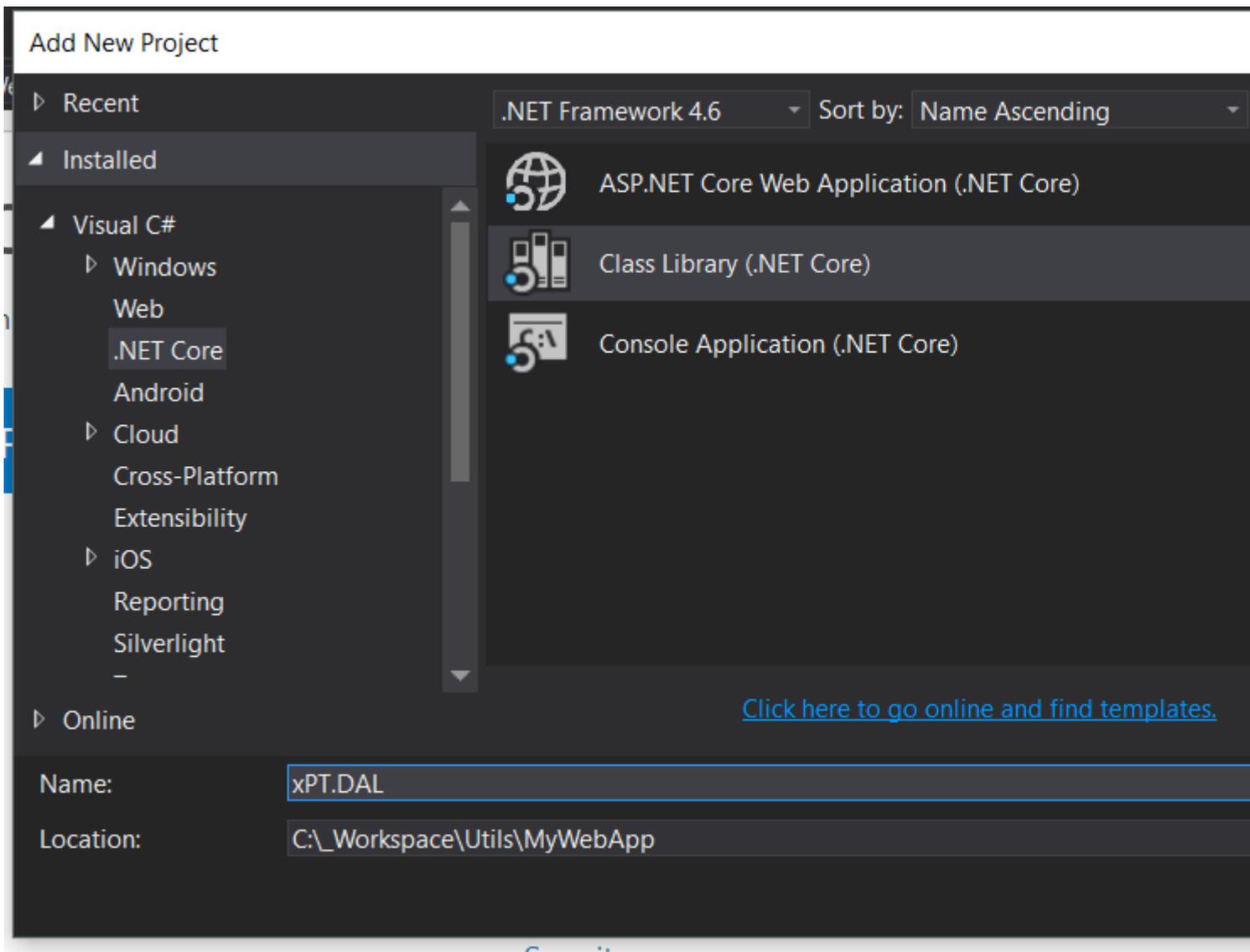


Step 2 - Create The Projects

Create a new ASP.NET Core Web Application --> Then Select Web Application in the next screen



Add a Class Library (.NET Core) Project



Step 3 - Installing EF Packages

Open your `project.json` file of Class Library, and paste the following, then Save the file:

```
{
  "version": "1.0.0-*",

  "dependencies": {
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "NETStandard.Library": "1.6.0"
  },
  "tools": {
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
  },

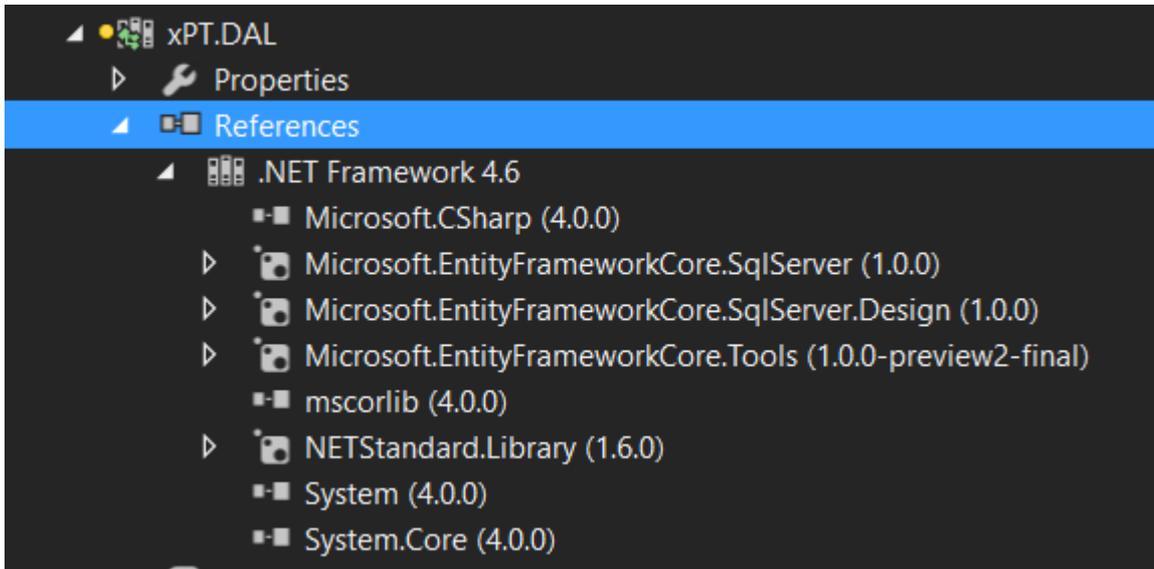
  "frameworks": {
    "net46": {
    },
    "netcoreapp1.0": {
      "dependencies": {
```

```

    "Microsoft.NETCore.App": {
      "type": "platform",
      "version": "1.0.0-*"
    }
  }
}
}
}
}
}

```

This should restore the packages under `References`



----- OR

You can install them using Nuget Package Manager by running the following commands in the Package Manager Console

```

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools -Pre

Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design

```

Note: Install one Package at a time - if you get an error after installing

```

Microsoft.EntityFrameworkCore.Tools

```

Then change the content of your `project.json` frameworks section to this:

```

"frameworks": {
  "net46": {
  },
  "netcoreapp1.0": {
    "dependencies": {
      "Microsoft.NETCore.App": {
        "type": "platform",
        "version": "1.0.0-*"
      }
    }
  }
}

```

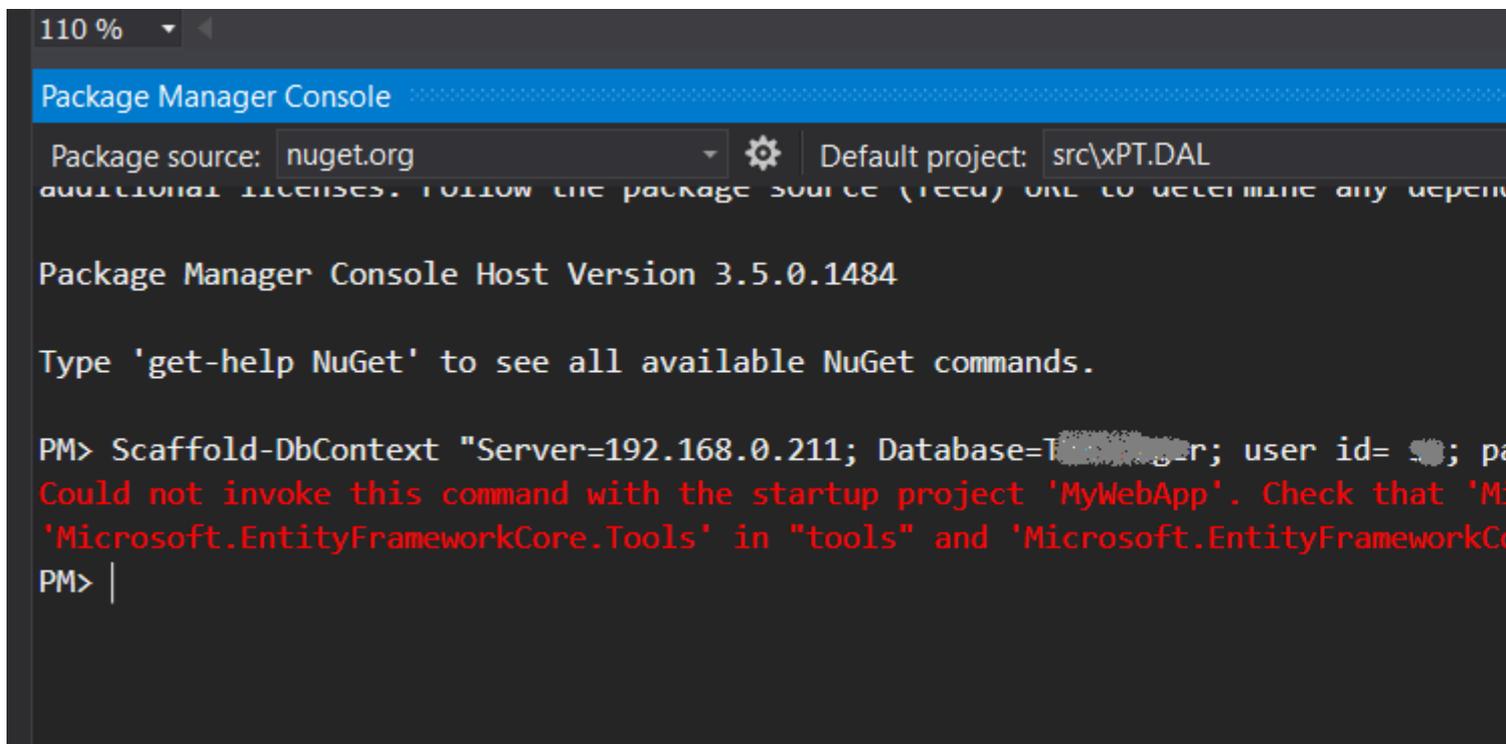
```
}  
}  
}  
}
```

Step 4 - Creating the Database Model

Now to generate the Database run the following command in the `Package Manager Console` (DON'T forget to Change the connection string to your Database)

```
Scaffold-DbContext "Server=. ; Database=DATABASE; user id= USER ; password = PASSWORD;"  
Microsoft.EntityFrameworkCore.SqlServer
```

This will give you the Error about Startup Project:



The screenshot shows the Package Manager Console window in Visual Studio. The console title is "Package Manager Console". The package source is set to "nuget.org" and the default project is "src\XPT.DAL". The console text includes: "Package Manager Console Host Version 3.5.0.1484", "Type 'get-help NuGet' to see all available NuGet commands.", and the command: "PM> Scaffold-DbContext "Server=192.168.0.211; Database=T...; user id= ...; pa...". The error message is: "Could not invoke this command with the startup project 'MyWebApp'. Check that 'Microsoft.EntityFrameworkCore.Tools' in 'tools' and 'Microsoft.EntityFrameworkCore.SqlServer' in 'dependencies' are installed in the startup project." The error message is displayed in red text.

For this you have to add the same references you added to Class Library to the .NET Web App

So open your `project.json` for the Web App,

Under `dependencies`, add:

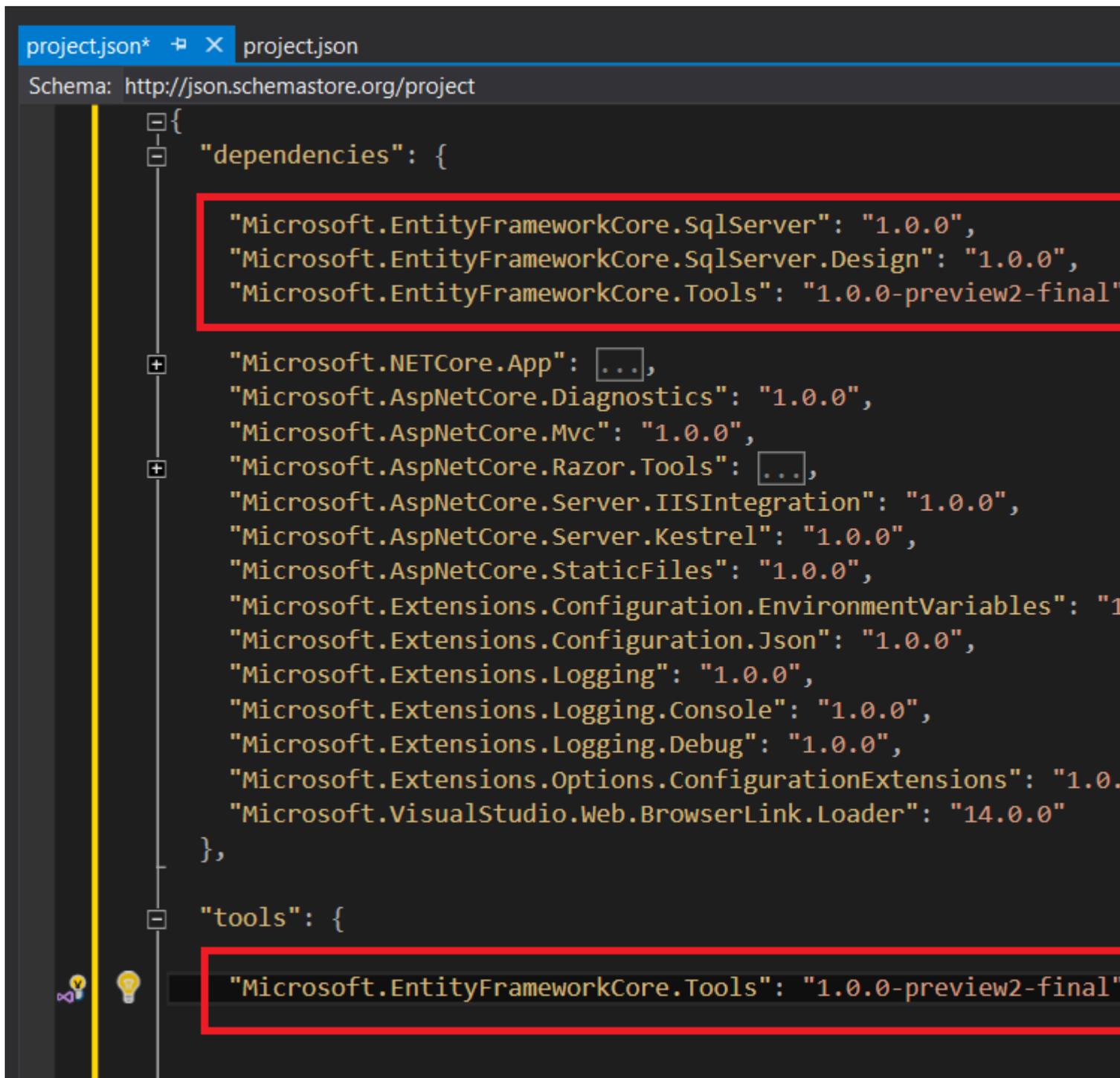
```
"Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",  
"Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",  
"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
```

and under `tools` add:

```
"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
```

After making the changes Save the file.

This is what my project.json looks like



```
project.json*  X project.json
Schema: http://json.schemastore.org/project
{
  "dependencies": {
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "Microsoft.NETCore.App": "...",
    "Microsoft.AspNetCore.Diagnostics": "1.0.0",
    "Microsoft.AspNetCore.Mvc": "1.0.0",
    "Microsoft.AspNetCore.Razor.Tools": "...",
    "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
    "Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
    "Microsoft.AspNetCore.StaticFiles": "1.0.0",
    "Microsoft.Extensions.Configuration.EnvironmentVariables": "1.0.0",
    "Microsoft.Extensions.Configuration.Json": "1.0.0",
    "Microsoft.Extensions.Logging": "1.0.0",
    "Microsoft.Extensions.Logging.Console": "1.0.0",
    "Microsoft.Extensions.Logging.Debug": "1.0.0",
    "Microsoft.Extensions.Options.ConfigurationExtensions": "1.0.0",
    "Microsoft.VisualStudio.Web.BrowserLink.Loader": "14.0.0"
  },
  "tools": {
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
  }
}
```

Then again run the command in Package Manager Console against the class library:

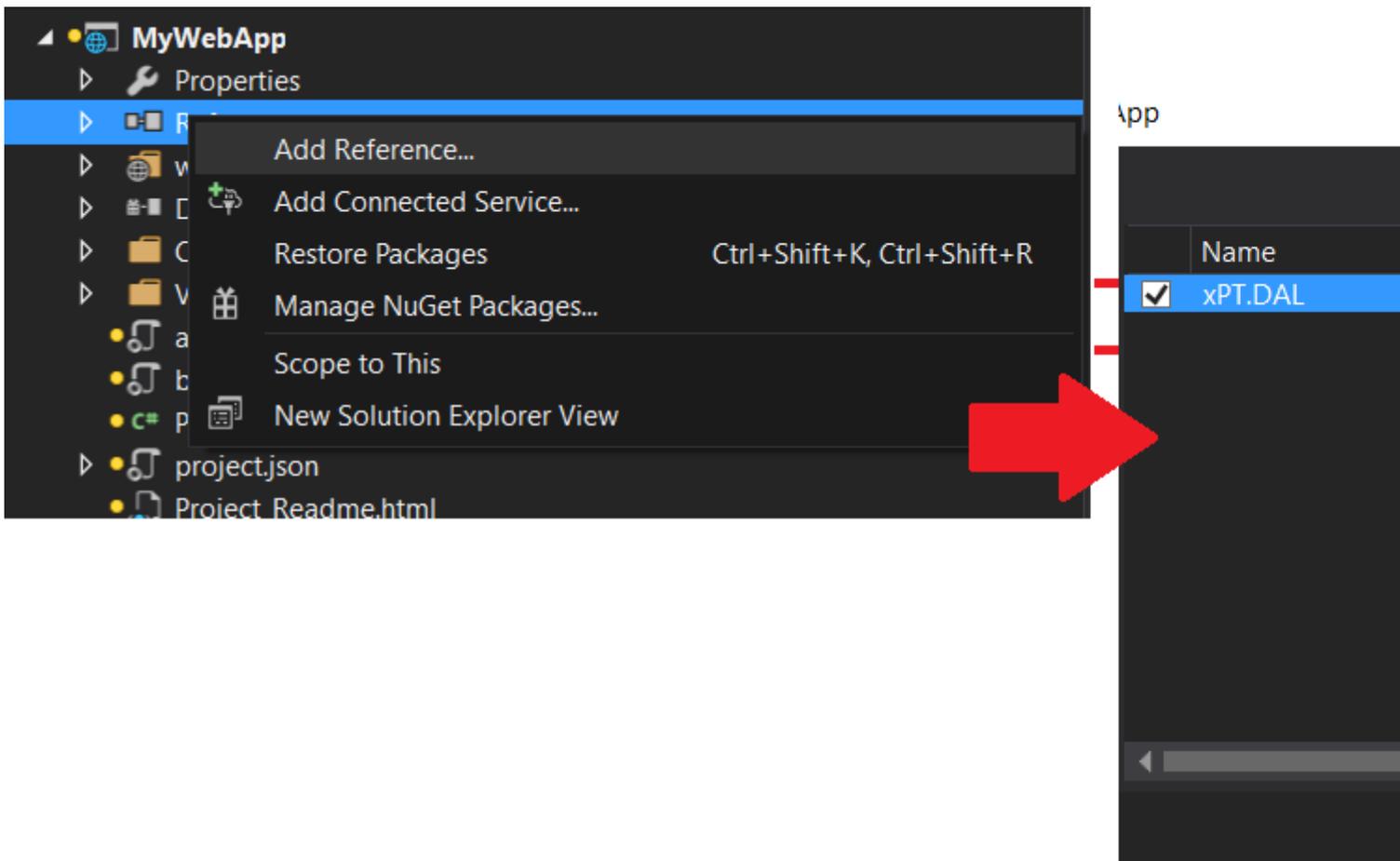
If you haven't already added the reference of your Class Library to the Web App, you will get this error:

```

PM> Scaffold-DbContext "Server=192.168.0.211; Database=
System.AggregateException: One or more errors occurred. (Could not find assembly
Microsoft.EntityFrameworkCore.Design.OperationOperationException: Could not find assembly
Microsoft.EntityFrameworkCore.Design.Internal.OperationExecutor..ctor(CommonOptio
    at Microsoft.EntityFrameworkCore.Tools.Cli.DbContextScaffoldCommand.<ExecuteA
--- End of inner exception stack trace ---
    at System.Threading.Tasks.Task.ThrowIfExceptional(Boolean includeTaskCanceled
    at System.Threading.Tasks.Task`1.GetResultCore(Boolean waitCompletionNotifica

```

to solve this add reference of your class Library to your Web App:



Finally

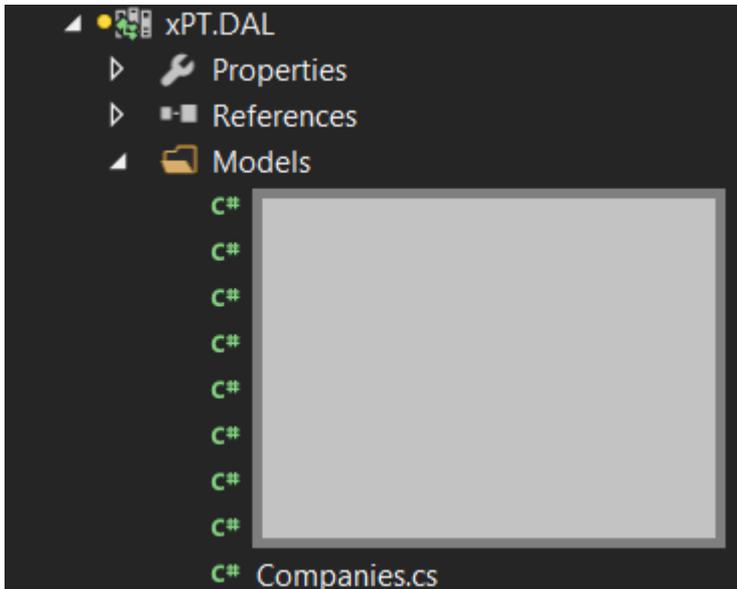
Run the Command again - in the Package Manager Console:

```

Scaffold-DbContext "Server=. ; Database=DATABASE; user id= USER ; password = PASSWORD;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

```

This should create the Entities under Models Folder, in the class library



Passing a Connection String

In my case here, we have a Multi Tenant Application, in which each client has their own Database, e.g. Client_1, Client_2, Client_3. So the connection string had to be dynamic.

So we added a connection string property to a constructor, and passed it to the Context in the `OnConfiguring` method

```
public partial class ClientContext
{
    private readonly string _connectionString;

    public ClientContext(string connectionString) : base()
    {
        _connectionString = connectionString;
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(_connectionString);
    }
}
```

and used it like this:

```
public void TestConnection()
{
    var clientId = 1;

    var connectionString = string.Format("Server=192.168.0.211; Database=Client_{0}; user id= USER; password = PWD;", clientId);

    using (var clientContext = new ClientContext(connectionString))
    {
        var assets = clientContext.Users.Where(s => s.UserId == 1);
    }
}
```

```
}
```

Model, Querying and Saving Data

Model

With EF Core, data access is performed using a model. A model is made up of entity classes and a derived context that represents a session with the database, allowing you to query and save data.

You can generate a model from an existing database, hand code a model to match your database, or use EF Migrations to create a database from your model (and evolve it as your model changes over time).

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace Intro
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=MyDatabase;Trusted_Connection=True");
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

Querying

Instances of your entity classes are retrieved from the database using Language Integrated Query (LINQ).

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

Saving Data

Data is created, deleted, and modified in the database using instances of your entity classes.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

Deleting Data

Instances of your entity classes are retrieved from the database using Language Integrated Query (LINQ).

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Attach(blog);
    db.Blogs.Remove(blog);
    db.SaveChanges();
}
```

Updating Data

Data is updated in the database using instances of your entity classes.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    var entity = db.Blogs.Find(blog);
    entity.Url = "http://sample2.com";
    db.SaveChanges();
}
```

Read **Getting started with Entity Framework Core** online: <https://riptutorial.com/entity-framework-core/topic/3796/getting-started-with-entity-framework-core>

Chapter 2: EF Core vs EF6.x

Remarks

For latest updates, please refer to: [Feature Comparison](#)

Examples

Side-by-side comparison

The following table compares the features available(1) in EF Core and EF6.x.

It is intended to give a high level comparison and does not list every feature, or attempt to give details on possible differences between how the same feature works.

Creating a Model	EF6.x	EF Core 1.0.0
Basic modelling (classes, properties, etc.)	Yes	Yes
Conventions	Yes	Yes
Custom conventions	Yes	Partial
Data annotations	Yes	Yes
Fluent API	Yes	Yes
Inheritance: Table per hierarchy (TPH)	Yes	Yes
Inheritance: Table per type (TPT)	Yes	
Inheritance: Table per concrete class (TPC)	Yes	
Shadow state properties		Yes
Alternate keys		Yes
Many-to-many: With join entity	Yes	Yes
Many-to-many: Without join entity	Yes	
Key generation: Database	Yes	Yes
Key generation: Client		Yes
Complex/value types	Yes	
Spatial data	Yes	

Creating a Model	EF6.x	EF Core 1.0.0
Graphical visualization of model	Yes	
Graphical drag/drop editor	Yes	
Model format: Code	Yes	Yes
Model format: EDMX (XML)	Yes	
Reverse engineer model from database: Command line		Yes
Reverse engineer model from database: VS wizard	Yes	
Incremental update model from database	Yes	

Querying Data	EF6.x	EF Core 1.0.0
LINQ: Simple queries	Stable	Stable
LINQ: Moderate queries	Stable	Stabilizing
LINQ: Complex queries	Stable	In-Progress
LINQ: Queries using navigation properties	Stable	In-Progress
“Pretty” SQL generation	Poor	Yes
Mixed client/server evaluation		Yes
Loading related data: Eager	Yes	Yes
Loading related data: Lazy	Yes	
Loading related data: Explicit	Yes	
Raw SQL queries: Model types	Yes	Yes
Raw SQL queries: Un-mapped types	Yes	
Raw SQL queries: Composing with LINQ		Yes

Saving Data	EF6.x	EF Core 1.0.0
SaveChanges	Yes	Yes
Change tracking: Snapshot	Yes	Yes
Change tracking: Notification	Yes	Yes
Accessing tracked state	Yes	Partial

Saving Data		EF6.x	EF Core 1.0.0
Optimistic concurrency		Yes	Yes
Transactions		Yes	Yes
Batching of statements			Yes
Stored procedure		Yes	
Detached graph support (N-Tier): Low level APIs		Poor	Yes
Detached graph support (N-Tier): End-to-end			Poor
Other Features		EF6.x	EF Core 1.0.0
Migrations		Yes	Yes
Database creation/deletion APIs		Yes	Yes
Seed data		Yes	
Connection resiliency		Yes	
Lifecycle hooks (events, command interception, ...)		Yes	
Database Providers	EF6.x	EF Core 1.0.0	
SQL Server	Yes	Yes	
MySQL	Yes	Paid only, unpaid coming soon (2)	
PostgreSQL	Yes	Yes	
Oracle	Yes	Paid only, unpaid coming soon (2)	
SQLite	Yes	Yes	
SQL Compact	Yes	Yes	
DB2	Yes	Yes	
InMemory (for testing)		Yes	
Azure Table Storage		Prototype	
Redis		Prototype	
Application Models	EF6.x	EF Core 1.0.0	
WinForms	Yes	Yes	

Application Models	EF6.x	EF Core 1.0.0
WPF	Yes	Yes
Console	Yes	Yes
ASP.NET	Yes	Yes
ASP.NET Core		Yes
Xamarin		Coming soon (3)
UWP		Yes

Footnotes:

(1) : As of 2016/10/18

(2) : Paid providers are available, unpaid providers are being worked on. The teams working on the unpaid providers have not shared public details of timeline etc.

(3) : EF Core is built to work on Xamarin when support for .NET Standard is enabled in Xamarin.

Read EF Core vs EF6.x online: <https://riptutorial.com/entity-framework-core/topic/7513/ef-core-vs-ef6-x>

Chapter 3: Updating a Many to Many relationship

Introduction

How to update a Many to Many relationship in EF Core:

Examples

MVC POST Edit example

Say we have a Product class with Multiple Colors which can be on many Products.

```
public class Product
{
    public int ProductId { get; set; }
    public ICollection<ColorProduct> ColorProducts { get; set; }
}

public class ColorProduct
{
    public int ProductId { get; set; }
    public int ColorId { get; set; }

    public virtual Color Color { get; set; }
    public virtual Product Product { get; set; }
}

public class Color
{
    public int ColorId { get; set; }
    public ICollection<ColorProduct> ColorProducts { get; set; }
}
```

Using this extension to make it easier:

```
public static class Extensions
{
    public static void TryUpdateManyToMany<T, TKey>(this DbContext db, IEnumerable<T>
currentItems, IEnumerable<T> newItems, Func<T, TKey> getKey) where T : class
    {
        db.Set<T>().RemoveRange(currentItems.Except(newItems, getKey));
        db.Set<T>().AddRange(newItems.Except(currentItems, getKey));
    }

    public static IEnumerable<T> Except<T, TKey>(this IEnumerable<T> items, IEnumerable<T>
other, Func<T, TKey> getKeyFunc)
    {
        return items
            .GroupJoin(other, getKeyFunc, getKeyFunc, (item, tempItems) => new { item,
tempItems })
    }
}
```

```

        .SelectMany(t => t.tempItems.DefaultIfEmpty(), (t, temp) => new { t, temp })
        .Where(t => ReferenceEquals(null, t.temp) || t.temp.Equals(default(T)))
        .Select(t => t.t.item);
    }
}

```

Updating a product's colors would look like this (a MVC Edit POST Method)

```

[HttpPost]
public IActionResult Edit(ProductVm vm)
{
    if (ModelState.IsValid)
    {
        var model = db.Products
            .Include(x => x.ColorProducts)
            .FirstOrDefault(x => x.ProductId == vm.Product.ProductId);

        db.TryUpdateManyToMany(model.ColorProducts, vm.ColorsSelected
            .Select(x => new ColorProduct
            {
                ColorId = x,
                ProductId = vm.Product.ProductId
            }), x => x.ColorId);

        db.SaveChanges();

        return RedirectToAction("Index");
    }
    return View(vm);
}

public class ProductVm
{
    public Product Product { get; set; }

    public IEnumerable<int> ColorsSelected { get; set; }
}

```

Code has been simplified as much as i can, no extra properties on any classes.

Read Updating a Many to Many relationship online: <https://riptutorial.com/entity-framework-core/topic/9527/updating-a-many-to-many-relationship>

Credits

S. No	Chapters	Contributors
1	Getting started with Entity Framework Core	Community , Dawood Awan , Dmitry , hasan , natemcmaster , NovaDev , tmg , uTeisT
2	EF Core vs EF6.x	Frédéric , Ruud Lenders , uTeisT
3	Updating a Many to Many relationship	Paw Ormstrup Madsen