

Exercise 5a : Entity Framework Code First

Objective

Entity Framework is an essential part of most .NET /CORE applications and is certainly used extensively in this course. We have this exercise to refresh/familiarise with Entity Framework.

This exercise will take around 45 minutes.

Referenced material

This exercise is based on material from the chapter " Entity Framework Code First".

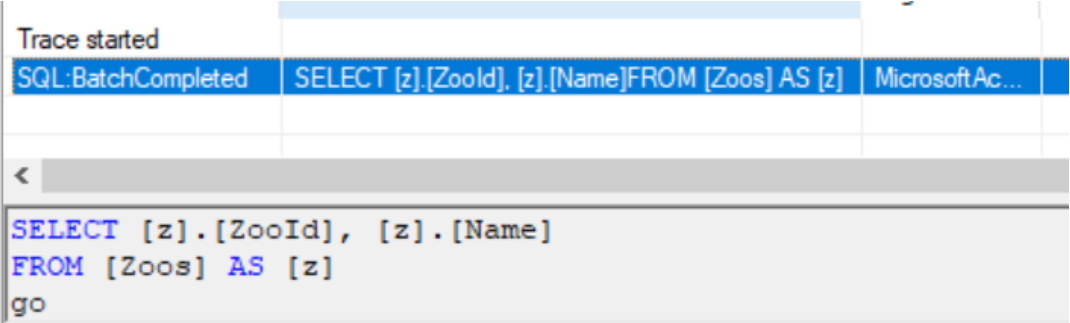
The 'Begin'

1	<p>Open the 'Begin' solution in Visual Studio and compile (Shift Ctrl+B).</p> <p>Have a look at the code and note: Program/Main(): Currently the program does not touch a database DataClasses: We have placed the data classes all in the 1 file. This is purely so you can see all the components. Operationally stick to 1 class in 1 file</p> <p>In order to focus on only the Entity Framework elements, the exercise is presented in a Console Application, but we will use Dependency Injection.</p> <p>There is nothing in the 'Begin' that wouldn't have been there even if you had no intention of storing this in a database. There are quite a few commented-out items</p> <p>Run it (Ctrl-F5) – you should get this:</p> <pre>Name = London, number of animals = 3 ...Elephant Dumbo ...Elephant Heffalumps ...Lion Clarence Name = Edinburgh, number of animals = 2 ...Panda Sweetie ...Panda Sunshine</pre> <p>We now proceed to add what you need to implement Entity Framework, starting with the code (ie Code First). Later, we will do this from an existing database (aka CodeFirstFromDatabase)</p>
2	<p>In Visual Studio, select View/Task List. This will show a window where you can see all the To Do comments that we've added to the code.</p> <p>Uncomment TODO 1 and confirm that the backpointer (animal to zoo) is null so you get a null reference exception.</p> <p>Re-instate the comment-out</p>

NuGet

3	Read TODO 2 and Nuget the packages listed.
4	We will need a context to describe which tables we wish to have in the database. Go to TODO 3 and uncomment the zoocontext class (but OnConfiguring still commented out. You will need to add a using clause for Microsoft.EntityFrameworkCore to the top of the file listing.
5	<i>Everything</i> in EF is a property – we’ve done this already. However, because we are going to store Zoo and Animal objects in a relational database, they must have an Id. Go to TODO 4 and uncomment these.
6	Comment out the foreach loop in Program/Main() and store it temporarily in Notepad
7	Delete the entire contents of Program/Main() and replace with <pre>static void Main(string[] args) { ConfigureServices(); Configure(); }</pre> <p>Hopefully these names should look familiar to you... Dependency Injection!</p>
8	Right-click ConfigureServices() > Quick Actions and refactoring > Generate Method. Repeat for Configure()
9	Paste your commented-out foreach loop (in Notepad) into Configure(), in place of the “throw new NotImplementedException” line. For now leave it commented out
10	Add in the following code – this is the Console equivalent of what ASPNET Core largely does for you : <pre>class Program { public static ServiceCollection services; public static ServiceProvider serviceProvider; static void Main(string[] args) { ConfigureServices(); Configure(); } private static void ConfigureServices() { services = new ServiceCollection(); string conn = @"Server=.\SqlExpress;Encrypt=False;Database=EFRefresh;Trusted_Connection=true; MultipleActiveResultSets=True"; services.AddDbContext<ZooContext>(options => options.UseSqlServer(conn)); serviceProvider = services.BuildServiceProvider(); } }</pre> <p>NB – in the above we have had to split the connection string across 2 lines to fit into the Word document. The Connection string must be all on 1 line, not just in this lab but in any other labs that follow</p>
11	In Configure() we use the injected ZooContext. Add this line: <pre>private static void Configure() { ZooContext ctx = serviceProvider.GetService<ZooContext>(); //foreach (Zoo zoo in zoos) //{</pre>

```
// Console.WriteLine($"\\nName = {zoo.Name}, number of animals =
{zoo.Animals.Count()}");
//     foreach (Animal animal in zoo.Animals)
//     {
//         Console.WriteLine($"...{animal.Type,-10}{animal.Name}");
//         // TODO 1 Console.WriteLine($".....in zoo {animal.Zoo.Name}");
//     }
// }
```

12	<p>Uncomment the foreach loop and make this 1 change:</p> <pre>foreach (Zoo zoo in ctx.Zoos)</pre>
13	<p>If we ran it now, it would fail because there is no database (if you're familiar with the older Entity Framework 6, this behaves differently – it would create the database automatically). In a Console App, the way that our database server is configured means that we have the permissions to be able to drop and create databases and hence seed some data. Note a web app does <i>not</i> typically have these permissions</p> <p>Add this code</p> <pre>ZooContext ctx = serviceProvider.GetService<ZooContext>(); ctx?.Database.EnsureDeleted(); ctx?.Database.EnsureCreated(); AddSampleData(ctx); foreach (Zoo zoo in ctx.Zoos)</pre> <p>Uncomment the provided code for AddSampleData() (TO DO 5)</p>
11	<p>Ctrl+F5 to run it and you should get the same output as before. Now comment out the three highlighted lines above you have just added in – the database is present and populated so we should get the same result when we run it? Try it and see. Can you explain why it worked with a new database but not an existing one? (Hint – Fix Up)</p>
12	<p>Let see what SQL commands EntityFramework is invoking. In the Assets folder is the Express Profiler. There is a .msi file but you can just run the .exe Press the green arrow to start a new trace and run the Console App (with the 3 lines commented out)</p> <p>The trace should show</p>  <p>I.e. only the Zoos have been requested.</p>
13	<p>Solve the problem by using Eager Loading</p> <pre>foreach (Zoo zoo in ctx.Zoos.Include(z=>z.Animals))</pre>
14	<p>Note we still have the 'backpointer' line <code>Console.WriteLine(\$".....in zoo {animal.Zoo.Name}");</code> Which previously caused a NullReferenceException. Confirm this now works – ie EntityFramework populates it for free.</p>
15	<p>Undo your Eager Loading solution. Solve the problem by Lazy Loading :</p>

	<p>Uncomment TODO 6 And use NuGet to import Microsoft.EntityFrameworkCore.Proxies Finally, make this change in the Zoo class</p> <pre>public virtual ICollection<Animal> Animals { get; set; } = new List<Animal>();</pre> <p>Run it - it should work. Use the profiler to see the 3 SQL statements</p>
16	<p>In OnConfiguration(), comment out <code>optionsBuilder.UseLazyLoadingProxies();</code></p>
17	<p>Explicit Loading is not so easy to remember. Implement it:</p> <pre>ctx.Entry(zoo).Collection(z=>z.Animals).Load(); Console.WriteLine(\$"{z.Name}..."); foreach (Animal animal in zoo.Animals)</pre> <p>Check it works</p>

Migration

18	<p>Make this change to the Animal class</p> <pre>public string Name { get; set; } public double Weight { get;set;} public virtual Zoo Zoo { get;set;}</pre> <p>Ctrl+F5 and confirm you get an exception:</p> <pre>Name = London, number of animals = 0 Unhandled Exception: System.Data.SqlClient.SqlException: Invalid column name 'Weight'. at System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnect</pre> <p>Comment out this new line.</p>
19	<p>We will use 'Migrations' to resolve this.</p> <p>Since Migrations does not use dependency injection, we need to tell it how to configure our connection. Uncomment the ZooContextFactory at the bottom of the ZooContext.cs file. Since this class implements the interface IDesignTimeDbContextFactory<ZooContext>, Migrations will automatically use it to create the context.</p>
20	<p>Now, select Tools/Nuget Package Manager/Package Manager Console</p> <p>Into the new console window that appears, type:</p> <pre>add-migration Initial</pre> <p>A class gets created (and opened) in the new Migrations folder, called Initial. It has a method called "Up", which creates the database (without the Weight property), but we don't need that to happen – our database already exists.</p> <p>Modify the method so that it doesn't do anything:</p> <pre>protected override void Up(MigrationBuilder migrationBuilder) { return; migrationBuilder.CreateTable(.....</pre> <p>Now, back in the package manager console, type:</p> <pre>update-database</pre> <p>And then, remove the "return" from the "Up" method. (This method won't run again, since entity framework remembers that this migration has already been run – unless we drop and create the database.)</p>
21	<p>Un-comment the Weight property. Then, type the following two commands in the package manager console to add it to the database:</p> <pre>add-migration AddWeight update-database</pre> <p>The new column has now been added to the Animals table. Check this is SSMS, and also press F5 and check your program now runs again.</p>
22	<p>All the animals have weights which have defaulted to 0. We could now easily modify our program to include weights for each animal and display those weights. If you have time, try to do this!</p>

