.NET 9 App Dev Hands-On Lab

EF Lab 3 – DbContext, EF Core Migrations

This lab creates the `DbContext` and the `DbContextFactory` as well as creates and executes your first migration. Before starting this lab, you must have completed EF Lab 2. The lab works on the `AutoLot.Dal` project. Begin by renaming the generated `Class1.cs` file to `GlobalUsings.cs`, and replace the scaffolded code with the following:

```
global using AutoLot.Models.Entities;
global using AutoLot.Models.Entities.Base;
global using AutoLot.Models.Entities.Configuration;
global using AutoLot.Models.ViewModels;
global using AutoLot.Models.ViewModels.Configuration;

global using Microsoft.Data.SqlClient;
global using Microsoft.EntityFrameworkCore;
global using Microsoft.EntityFrameworkCore.ChangeTracking;
global using Microsoft.EntityFrameworkCore.Design;
global using Microsoft.EntityFrameworkCore.Diagnostics;
global using Microsoft.EntityFrameworkCore.Metadata;
global using Microsoft.EntityFrameworkCore.Migrations;
global using Microsoft.EntityFrameworkCore.Query;
global using Microsoft.EntityFrameworkCore.Storage;
global using Microsoft.Extensions.DependencyInjection;

global using System.Data;
global using System.Linq.Expressions;
```

Part 1: Create the derived DbContext Class

The derived `DbContext` class is the hub for using EF Core with C#.

Step 1: Create the ApplicationDbContext.cs file and its Constructor

- Create a new folder named `EfStructures` in the `AutoLot.Dal` project. Add a new class to the folder named `ApplicationDbContext.cs`. Make the class `public`, add the default constructor, and the `DbSet<T>` properties:

```
namespace AutoLot.Dal.EfStructures;
public class ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
 : DbContext(options)
{
  public DbSet<Car> Cars { get; set; }
  public DbSet<Driver> Drivers { get; set; }
  public DbSet<CarDriver> CarsToDrivers { get; set; }
  public DbSet<Make> Makes { get; set; }
  public DbSet<Radio> Radios { get; set; }
  public DbSet<SeriLogEntry> SeriLogEntries { get; set; }
}
```

Step 2: Add the OnModelCreating method and Register the Configuration Classes

- Add the override for `OnModelCreating` into the `ApplicationDbContext.cs` class. The Fluent API code is placed in this method, and the configuration classes are registered. Add the following:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  new CarConfiguration().Configure(modelBuilder.Entity<Car>());
  new DriverConfiguration().Configure(modelBuilder.Entity<Driver>());
  new CarDriverConfiguration().Configure(modelBuilder.Entity<CarDriver>());
  new RadioConfiguration().Configure(modelBuilder.Entity<Radio>());
  new MakeConfiguration().Configure(modelBuilder.Entity<Make>());
  new SeriLogEntryConfiguration().Configure(modelBuilder.Entity<SeriLogEntry>());
  new CarViewModelConfiguration().Configure(modelBuilder.Entity<CarViewModel>());
}
```

Step 3: Update the GlobalUsings.cs file

- Add the following to the `GlobalUsings.cs` file:

```
global using AutoLot.Dal.EfStructures;
```

Part 2: Create the ApplicationDbContextFactory Class

The `IDesignTimeDbContextFactory` is used by the design-time tools and the CLI to instantiate a new instance of the `ApplicationDbContext`.

- Add a new class named `ApplicationDbContextFactory.cs` to the `EfStructures` folder and update the code to the following: **NOTE:** Update your connection string to fit your environment.

```
namespace AutoLot.Dal.EfStructures;
public class ApplicationDbContextFactory : IDesignTimeDbContextFactory<ApplicationDbContext>
{
  public ApplicationDbContext CreateDbContext(string[] args)
  {
    var optionsBuilder = new DbContextOptionsBuilder<ApplicationDbContext>();
    var connectionString =
        @"server=(localdb)\MsSqlLocalDb;Database=AutoLot_Hol;Integrated Security=true";
    //var connectionString =
    //  @"server=(localdb)\ProjectModels;Database=AutoLot_Hol;Trusted_Connection=True;";
    //var connectionString =
    //  @"server=.,5433;Database=AutoLot_Hol;User Id=sa;Password=P@ssw0rd;Encrypt=false;";
    optionsBuilder.UseSqlServer(connectionString);
    optionsBuilder.ConfigureWarnings(cw => cw.Ignore(RelationalEventId.BoolWithDefaultWarning));
    Console.WriteLine(connectionString);
    return new ApplicationDbContext(optionsBuilder.Options);
  }
}
```

- If using anything other than `LocalDb`, you must disable encryption by adding this to the connection string:

```
Encrypt=false;
```

Part 3: Update the Database Using EF Core Migrations

Migrations are created and executed using the .NET Core EF Command Line Interface. The commands must be executed from the same directory as the `AutoLot.Dal.csproj` file.

If the `Microsoft EntityFrameworkCore.Tools` package was installed, NuGet-style commands can be used in the Package Manager Console in Visual Studio. The commands shown here are the .NET CLI version.

Step 1: Install/Update the EF Core CLI Global Tool

- Run the following command if you have installed a previous EF Core Global Tool version to uninstall it:

```
dotnet tool uninstall --global dotnet-ef
```

- Run the following command to install the EF Core Global Tooling version 9.0:

```
dotnet tool install --global dotnet-ef --version 9.0.0
```

- Alternately, you can update the tooling to the latest version (including pre-release versions) with the following command:

```
dotnet tool update --global dotnet-ef –prerelease
```

- You can test the install by typing the following (if successful, you will see the EF Unicorn in ASCII art):

```
dotnet ef
```

- You can check on the version history at NuGet.org:

```
https://www.nuget.org/packages/dotnet-ef
```

Step 2: Create and Execute the Initial Migration

- Open a command prompt/PS window in the same directory as the `AutoLot.Dal` project
  OR
  [Visual Studio] Open Package Manager Console (Ctrl Q -> Package Manager Console) and navigate to the `AutoLot.Dal` project directory.
- Create the initial migration with the following command (-o = output directory, -c = Context File):
  Note: In this project, specifying the context is optional since there is only one context/context factory in the database

```
[Windows]
```

NOTE: The following lines must be entered as one line – copying and pasting from this document doesn’t work without removing the line break

```
dotnet ef migrations add Initial -o EfStructures\Migrations -c
AutoLot.Dal.EfStructures.ApplicationDbContext
```

NOTE: The above lines must be entered as one line – copying and pasting from this document doesn’t work without removing the line break

```
[Non-Windows]
```

NOTE: The following lines must be entered as one line – copying and pasting from this document doesn’t work without removing the line break

```
dotnet ef migrations add Initial -o EfStructures/Migrations -c
AutoLot.Dal.EfStructures.ApplicationDbContext
```

NOTE: The above lines must be entered as one line – copying and pasting from this document doesn’t work without removing the line break

- This creates three files in the `EfStructures\Migrations (EfStructures/Migrations)` Directory:

```
A file named YYYYMMDDHHmmSS_Initial.cs (where date time is UTC)
A file named YYYYMMDDHHmmSS _Initial.Designer.cs (same numbers)
ApplicationDbContextModelSnapshot.cs
```

- Open the `YYYYMMDDHHmmSS _Initial.cs` file. Check the Up and Down methods to make sure the database and table/column creation code is there
- Update the database with the following command (specifying the context is optional since there is only one context in the project):

```
dotnet ef database update Initial -c AutoLot.Dal.EfStructures.ApplicationDbContext
```

- Examine your database in SQL Server Management Studio\Azure Data Studio, or Visual Studio, to ensure the tables were created.

Summary

In this lab, you created the `ApplicationDbContext` and the `ApplicationDbContextFactory`. The final step was to create the initial migration and update the database.

Next steps

In the next part of this tutorial series, you will create the SQL Server objects, including a stored procedure, two views, and a user-defined function.