Build an EF and ASP.NET Core 3.0 App HOL

Lab 5

This lab walks you through creating the Data Initializer. Prior to starting this lab, you must have completed Lab 4.

Step 1: Create the Sample Data

1) Create a new folder named Initialization in the AutoLot.Dal project. In this folder, add a new class named SampleData.cs. Add the following using statements to the top of the file:

```
using System.Collections.Generic;
using AutoLot.Models.Entities;
using AutoLot.Models.Entities.Owned;
```

2) Make the class static, and add in all of the lists for the sample data:

```
public static class SampleData
{
  public static List<Customer> Customers =>  new List<Customer>
  {
    new Customer
      {Id = 1, PersonalInformation = new Person{FirstName = "Dave", LastName = "Brenner"}},
    new Customer
      {Id = 2, PersonalInformation = new Person{FirstName = "Matt", LastName = "Walton"}},
    new Customer
      {Id = 3, PersonalInformation = new Person{FirstName = "Steve", LastName = "Hagen"}},
    new Customer
      {Id = 4, PersonalInformation = new Person{FirstName = "Pat", LastName = "Walton"}},
    new Customer
      {Id = 5, PersonalInformation = new Person{FirstName = "Bad", LastName = "Customer"}},
    };
  public static List<Make> Makes => new List<Make>
  {
    new Make {Id = 1, Name = "VW"},
    new Make {Id = 2, Name = "Ford"},
    new Make {Id = 3, Name = "Saab"},
    new Make {Id = 4, Name = "Yugo"},
    new Make {Id = 5, Name = "BMW"},
    new Make {Id = 6, Name = "Pinto"},
  };
  public static List<Car> Inventory => new List<Car>
  {
    new Car {Id = 1, MakeId = 1, Color = "Black", PetName = "Zippy"},
    new Car {Id = 2, MakeId = 2, Color = "Rust", PetName = "Rusty"},
    new Car {Id = 3, MakeId = 3, Color = "Black", PetName = "Mel"},
    new Car {Id = 4, MakeId = 4, Color = "Yellow", PetName = "Clunker"},
    new Car {Id = 5, MakeId = 5, Color = "Black", PetName = "Bimmer"},
    new Car {Id = 6, MakeId = 5, Color = "Green", PetName = "Hank"},
    new Car {Id = 7, MakeId = 5, Color = "Pink", PetName = "Pinky"},
    new Car {Id = 8, MakeId = 6, Color = "Black", PetName = "Pete"},
    new Car {Id = 9, MakeId = 4, Color = "Brown", PetName = "Brownie"},
  };
```

```
  public static List<Order> Orders => new List<Order>
  {
    new Order {Id = 1, CustomerId = 1, CarId = 5},
    new Order {Id = 2, CustomerId = 2, CarId = 1},
    new Order {Id = 3, CustomerId = 3, CarId = 4},
    new Order {Id = 4, CustomerId = 4, CarId = 7},
  };

  public static List<CreditRisk> CreditRisks => new List<CreditRisk>
  {
    new CreditRisk
    {
      Id = 1,
      CustomerId = Customers[4].Id,
      PersonalInformation = new Person
      {
        FirstName = Customers[4].PersonalInformation.FirstName,
        LastName = Customers[4].PersonalInformation.LastName
      }
    }
  };
}
```

Step 2: Create the Sample Data Initializer

1) In the Initialization folder, add a new class named SampleDataInitializer.cs. Add the following using statements to the top of the file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using AutoLot.Dal.EfStructures;
using AutoLot.Models.Entities;
using AutoLot.Models.Entities.Base;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Storage;
```

2) Add a method to clear all of the data from the database:

```
internal static void ClearData(ApplicationDbContext context)
{
  var entities = new[] {
                typeof(Order).FullName,
                typeof(Customer).FullName,
                typeof(Car).FullName,
                typeof(Make).FullName,
                typeof(CreditRisk).FullName
  };
  foreach (var entityName in entities)
  {
    var entity = context.Model.FindEntityType(entityName);
    var tableName = entity.GetTableName();
    var schemaName = entity.GetSchema();
    context.Database.ExecuteSqlRaw($"DELETE FROM {schemaName}.{tableName}");
    context.Database.ExecuteSqlRaw($"DBCC CHECKIDENT (\"{schemaName}.{tableName}\", RESEED, 1);");
  }
}
```

3) Add a method to insert a list of entities into the database. This method creates an execution strategy and a database transaction to process the data:

```
internal static void ProcessInsert<TEntity>(ApplicationDbContext context, DbSet<TEntity> table,
    List<TEntity> records) where TEntity : BaseEntity
{
  if (table.Any())
  {
    return;
  }

  IExecutionStrategy strategy = context.Database.CreateExecutionStrategy();
  strategy.Execute(() =>
  {
    using var transaction = context.Database.BeginTransaction();
    try
    {
      var metaData = context.Model.FindEntityType(typeof(TEntity).FullName);
      context.Database.ExecuteSqlRaw(
        $"SET IDENTITY_INSERT {metaData.GetSchema()}.{metaData.GetTableName()} ON");
      table.AddRange(records);
      context.SaveChanges();
      context.Database.ExecuteSqlRaw(
        $"SET IDENTITY_INSERT {metaData.GetSchema()}.{metaData.GetTableName()} OFF");
      transaction.Commit();
    }
    catch (Exception)
    {
      transaction.Rollback();
    }
  });
}
```

4) Add a method to call the ProcessInsert method using the sample data:

```
internal static void SeedData(ApplicationDbContext context)
{
  try
  {
    ProcessInsert(context, context.Customers, SampleData.Customers);
    ProcessInsert(context, context.Makes, SampleData.Makes);
    ProcessInsert(context, context.Cars, SampleData.Inventory);
    ProcessInsert(context, context.Orders, SampleData.Orders);
    ProcessInsert(context, context.CreditRisks, SampleData.CreditRisks);
  }
  catch (Exception ex)
  {
    Console.WriteLine(ex);
    throw;
  }
}
```

5) Add a method to drop and create the database along with the migrations:
NOTE: The `EnsureCreated` method builds the database from the entity model, but doesn't execute any migrations and blocks future migrations. The `Migrate` method creates the database (if needed) and runs all migrations for a DbContext.

```
internal static void DropAndCreateDatabase(StoreContext context)
{
  context.Database.EnsureDeleted();
  context.Database.Migrate();
}
```

6) The InitializeData method drops the database, rebuilds it, and seeds the sample data:

```
public static void InitializeData(ApplicationDbContext context)
{
  DropAndCreateDatabase(context);
  SeedData(context);
}
```

7) The ClearAndReseedDatabase method clears the data and then seeds the sample data.

```
public static void InitializeData(StoreContext context)
{
  ClearData(context);
  SeedData(context);
}
```

Summary

This lab created data initializer, completing the data access layer.

Next steps

In the next part of this tutorial series, you will add in integration tests for the data access layer.