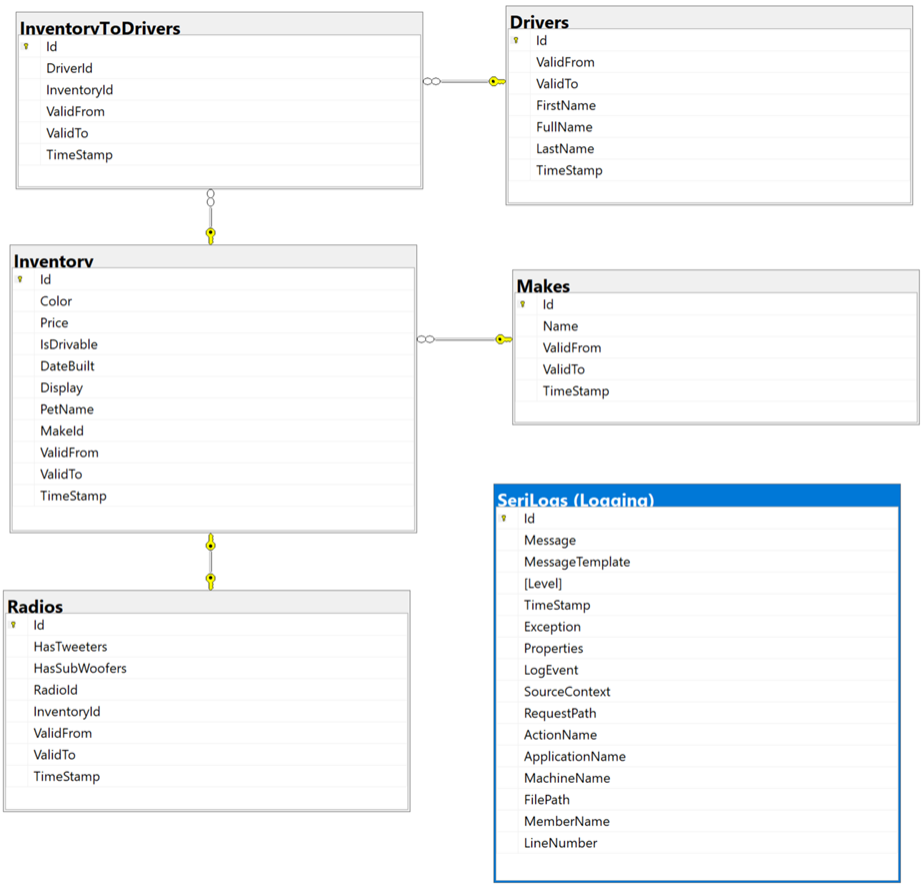
.NET 10 App Dev Hands-On Lab

EF Lab 2 – Entities and Models

This lab takes you through creating the `Models` and `ViewModels`. Before starting this lab, you must have completed EF Lab 1.

This lab will create the models that represent this database:



All work in this lab is to be completed in the `AutoLot.Models` project.

Part 1: The GlobalUsings.cs File

- Begin by renaming the autogenerated `Class1.cs` to `GlobalUsings.cs` and clearing out the templated code. Add the following `global using` statements to the file:

```
global using Microsoft.EntityFrameworkCore;
global using Microsoft.EntityFrameworkCore.Metadata.Builders;
global using System.ComponentModel;
global using System.ComponentModel.DataAnnotations;
global using System.ComponentModel.DataAnnotations.Schema;
global using System.Globalization;
global using System.Xml.Linq;
```

- You can also add global using statements into the project file. I find that this mechanism causes too much abstraction from the code. However, if you would like to try this method, here is an example:

```
<ItemGroup>
  <Using Include="Microsoft.EntityFrameworkCore" />
</ItemGroup>
```

Part 2: Creating the Entities and ViewModel in AutoLot.Models

The entities represent the data that is persisted in SQL Server. We use the Table Per Hierarchy (TPH) pattern for this workshop.

Copilot Agent Mode

```
Setup Prompt: Always use file scoped namespaces. Always combine attributes on a single line when
possible. The project does not use nullable reference types. There is a GlobalUsings.cs file that
includes common usings, don't include using statements in new files if they are already in the
globalusings.cs file.I prefer expression bodied members when possible. Single line if statements
should still use braces. Use ternary operators when appropriate.
Use internal over private. All classes and methods are public unless told otherwise. Don't add a
constructor unless instructed to do so. Use primary constructors when possible and don't declare a
class level variable if the parameter from the constructor can be used. Don't initialize
properties unless instructed to do so. Use MaxLength instead of StringLength for maximum length
attributes.

Prompt: In the autolot.models project, create a new folder named Entities. In that folder, create
another folder name Base.
Inside the base folder, create an abstract class named BaseEntity with the following public
properties:
Id (int, Attributes: Key, Database Generated Identity )
TimeStamp (long,Timestamp attribute).

Prompt: Add the following global usings to the GlobalUsings.cs file if they do not already exist
(sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Models.Entities;
global using AutoLot.Models.Entities.Base;
```

Prompt: In the Autolot.Models project Entities folder, create a new folder named ComplexTypes. In that folder, create a class named Person.
Add the ComplexType attribute to the class and add the following public properties:
FirstName (string, attributes: required, max length 50)
LastName (string, attributes: required, max length 50)
FullName (string, attributes: Database Generated and Computed)

Prompt: Add the following global usings to the GlobalUsings.cs file if they do not already exist (sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Models.Entities.ComplexTypes;

Prompt: In the Autolot.Models project Entities folder, create a class named Car and inherit from the BaseEntity.
Call the table name “Inventory” in the “dbo” schema.
Add an index to the class on MakeId named "IX_Inventory_MakeId"
Add the following public properties:
Color (string, attributes: required, max length 50)
Price (string)
IsDrivable (bool, initialized to true, attributes: DisplayName = "Is Drivable")
DateBuilt (DateTime?)
Display (string, attributes: Database Generated and Computed)
PetName (string, attributes: required, max length 50, DisplayName = “Pet Name”)
MakeId (int, attributes: required, DisplayName = “Make”)

Prompt: In the Autolot.Models project Entities folder, create a class named CarDriver and inherit from the BaseEntity. Call the table name “InventoryToDrivers” in the “dbo” schema. Add the following public properties:
DriverId (int)
CarId (int, attributes: Column = “InventoryId”)
Add a unique index to CarDriver.cs on the combination of DriverId and CarId named "IX_InventoryToDrivers_DriverId_CarId"

Prompt: In the Autolot.Models project Entities folder, create a class named Driver and inherit from the BaseEntity. Call the table name “Drivers” in the “dbo” schema. Add the following public properties:
PersonInformation (person, initialized to new Person)

Prompt: In the Autolot.Models project Entities folder, create a class named Make and inherit from the BaseEntity. Call the table name “Makes” in the “dbo” schema. Add the following public properties:
Name (string, attributes: required, max length 50)

Prompt: In the Autolot.Models project Entities folder, create a class named Radio and inherit from the BaseEntity. Call the table name “Radios” in the “dbo” schema. Add the following public properties:
HasTweeters (bool)
HasSubWoofers (bool)
RadioId (string, attributes: required, max length 50)
CarId (int, attributes: column name “InventoryId”)

Manual

Step 1: Create the Base Entity

- Create a new folder in the `AutoLot.Models` project named `Entities`. Create a subfolder named `Base` under the `Entities` folder. Add a new class to the `Base` folder named `BaseEntity.cs`, and update the code to the following:

```
namespace AutoLot.Models.Entities.Base;
public abstract class BaseEntity
{
  [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
  public int Id { get; set; }
  [Timestamp]
  public long TimeStamp { get; set; }
}
```

Step 2: Create the Person Complex Type Class

`ComplexType` classes can be reused between other entities and folded into the parent entity's table.

- Add a `ComplexTypes` folder under the `Entities` folder and a new class named `Person.cs`. Update the code for the class to the following:

```
namespace AutoLot.Models.Entities.ComplexTypes;
[ComplexType]
public class Person
{
  [Required, StringLength(50)]
  public string FirstName { get; set; }
  [Required, StringLength(50)]
  public string LastName { get; set; }
  [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
  public string FullName { get; set; }
}
```

- Add the following global using statement to the `GlobalUsings.cs` file:

```
global using AutoLot.Models.Entities;
global using AutoLot.Models.Entities.Base;
global using AutoLot.Models.Entities.ComplexTypes;
```

Step 3: Create the Car Entity

- Add a new class to the `Entities` folder named `Car.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;
[Table("Inventory", Schema = "dbo")]
[Index(nameof(MakeId), Name = "IX_Inventory_MakeId")]
public class Car : BaseEntity
{
  [Required,StringLength(50)]
  public string Color { get; set; }
  public string Price { get; set; }
  [DisplayName("Is Drivable")]
  public bool IsDrivable { get; set; } = true;
  public DateTime? DateBuilt { get; set; }
  [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
  public string Display { get; set; }
  [Required, StringLength(50), DisplayName("Pet Name")]
  public string PetName { get; set; }
  [Required, DisplayName("Make")]
  public int MakeId { get; set; }
}
```

Step 4: Create the CarDriver Entity

- Add a new class to the `Entities` folder named `CarDriver.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;
[Table("InventoryToDrivers", Schema = "dbo")]
[Index(nameof(DriverId), nameof(CarId), IsUnique = true, Name =
"IX_InventoryToDrivers_DriverId_CarId")]
public class CarDriver : BaseEntity
{
  public int DriverId { get; set; }
  [Column("InventoryId")]
  public int CarId { get; set; }
}
```

Step 5: Create the Driver Entity

- Add a new class to the `Entities` folder named `Driver.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;
[Table("Drivers", Schema = "dbo")]
public class Driver : BaseEntity
{
  public Person PersonInformation { get; set; } = new();
}
```


Step 6: Create the Make Entity

- Add a new class to the `Entities` folder named `Make.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;
[Table("Makes", Schema = "dbo")]
public class Make : BaseEntity
{
  [Required, StringLength(50)]
  public string Name { get; set; }
}
```

Step 7: Create the Radio Entity

- Add a new class to the `Entities` folder named `Radio.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;
[Table("Radios", Schema = "dbo")]
public class Radio : BaseEntity
{
  public bool HasTweeters { get; set; }
  public bool HasSubWoofers { get; set; }
  [Required, StringLength(50)]
  public string RadioId { get; set; }
  [Column("InventoryId")]
  public int CarId { get; set; }
}
```

Step 8: Create the Logging Entity

The `SeriLog` logging framework enables log entries to be written to a database table, in addition to text files, Application Insights, and many other targets. We will use EF Core to create the tables, even though they don't represent domain entities.

Copilot Agent Mode

```
Prompt: In the Autolot.Models project Entities folder, create a class named SeriLogEntry. Call the
table name "SeriLogs" in the "Logging" schema. Add the following public properties:
Id (int, attributes: Key, Database Generated Identity)
Message, MessageTemplate (string)
Level (string, attributes: max length 128)
TimeStamp (DateTime)
Exception, Properties, LogEvent, SourceContext, RequestPath, ActionName, ApplicationName,
MachineName, FilePath, MemberName (string)
LineNumber (nullable int)
PropertiesXml (XElement, readonly property with value of: Properties != null ?
XElement.Parse(Properties) : null, attributes: not mapped)
```

Manual

- Add a new class to the `Entities` folder named `SeriLogEntry.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;
[Table("SeriLogs", Schema = "Logging")]
public class SeriLogEntry
{
  [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
  public int Id { get; set; }
  public string Message { get; set; }
  public string MessageTemplate { get; set; }
  [MaxLength(128)]
  public string Level { get; set; }
  [DataType(DataType.DateTime)]
  public DateTime TimeStamp { get; set; }
  public string Exception { get; set; }
  public string Properties { get; set; }
  public string LogEvent { get; set; }
  public string SourceContext { get; set; }
  public string RequestPath { get; set; }
  public string ActionName { get; set; }
  public string ApplicationName { get; set; }
  public string MachineName { get; set; }
  public string FilePath { get; set; }
  public string MemberName { get; set; }
  public int? LineNumber { get; set; }
  [NotMapped]
  public XElement PropertiesXml => (Properties != null) ? XElement.Parse(Properties) : null;
}
```

Step 9: Create the CarViewModel in AutoLot.Models

Copilot Agent Mode

```
Prompt: In the Autolot.Models project, create a new folder named ViewModels.
In that folder, create a class named CarViewModel with the Keyless attribute and following public
properties:
Id (int)
IsDrivable (bool)
DateBuilt (DateTime?)
Price (string)
MakeId (int)
Color (string, initialized to string.Empty)
PetName (string, initialized to string.Empty)

Add the following global usings to the GlobalUsings.cs file if they do not already exist (sorted
alphabetically. Don't remove any existing global using statements).
global using AutoLot.Models.ViewModels;
```

Manual

- Create a new folder named `ViewModels` under the `AutoLot.Models` project. Add a new class named `CarViewModel.cs` into the `ViewModels` folder and update the code for the class to the following:

```
namespace AutoLot.Models.ViewModels;
[Keyless]
public class CarViewModel
{
  public int Id { get; set; }
  public bool IsDrivable { get; set; }
  public DateTime? DateBuilt { get; set; }
  public string Price { get; set; }
  public int MakeId { get; set; }
  public string Color { get; set; } = string.Empty;
  public string PetName { get; set; } = string.Empty;
}
```

Part 3: Creating the Navigation Properties

Navigation properties represent foreign key relationships between entities.

NOTE: The code will not successfully compile until this section is completed When updating classes, only add in the Bold code snippets.

Copilot Agent Mode

```
Setup Prompt: The project does not use nullable reference types. There is a GlobalUsings.cs file
that includes common usings, don't include using statements in new files if they are already in
the globalusings.cs file. I prefer expression bodied members when possible. Single line if
statements should still use braces. Use ternary operators when appropriate. Use internal over
private. All classes and methods are public unless told otherwise. Don't add a constructor unless
instructed to do so. Use primary constructors when possible and don't declare a class level
variable if the parameter from the constructor can be used. Don't initialize properties unless
instructed to do so.

Prompt: When creating collection navigation properties, always initialize them to new instances of
the collection as an List<T>.
Also, always use ICollection<T> for collection navigation properties.
Name navigation properties using plural names for collections and singular names with navigation
as the suffix for single instances.
Always use the nameof() function instead of string literals when specifying foreign key names and
inversepropeties in attributes.
Always add the ForeignKey atribute for intances navigation properties unless otherwise specified.
Always add the InverseProperty attribute for all navigation properties.

Prompt: In the Auolot.Models project, update Car.cs:
Add MakeNavigation, ForeignKey of MakeId, InverseProperty of Make.Cars
Add RadioNavigation, No ForeignKey, InverseProperty of Radio.CarNavigation
Add ICollection<Driver> Drivers, InverseProperty of Driver.Cars
Add ICollection<CarDriver> CarDrivers, InverseProperty of CarDriver.CarNavigation
Update CarDriver.cs:
Add DriverNavigation, ForeignKey of DriverId, InverseProperty of Driver.CarDrivers
Add CarNavigation, ForeignKey of CarId, InverseProperty of Car.CarDrivers
Update Driver.cs:
Add ICollection<Car> Cars, InverseProperty of Car.Drivers
Add ICollection<CarDriver> CarDrivers, InverseProperty of CarDriver.DriverNavigation
Update Make.cs:
Add ICollection<Car> Cars, InverseProperty of Car.MakeNavigation
Update Radio.cs:
Add CarNavigation, ForeignKey of CarId, InverseProperty of Car.RadioNavigation

Prompt: In the Autolot.Models project, update Car.cs with the following property:
Add public string MakeName (string, not mapped, readonly property with value of:
MakeNavigation?.Name ?? "Unknown")
Add override for ToString() method to return $"{PetName ?? "**No Name**"} is a {Color}
{MakeNavigation?.Name} with ID {Id}.".
```

This completes this section. The project will now build successfully. Confirm the generated code and then proceed to Part 4

Manual

Step 1: Update the Car Entity

- Update the `Car` entity by adding the following reference and collection navigation properties:

```
public class Car: BaseEntity
{
  //omitted for brevity

  [ForeignKey(nameof(MakeId))]
  [InverseProperty(nameof(Make.Cars))]
  public Make MakeNavigation { get; set; }
  [InverseProperty(nameof(Radio.CarNavigation))]
  public Radio RadioNavigation { get; set; }
  [InverseProperty(nameof(Driver.Cars))]
  public ICollection<Driver> Drivers { get; set; } = new List<Driver>();
  [InverseProperty(nameof(CarDriver.CarNavigation))]
  public ICollection<CarDriver> CarDrivers { get; set; } = new List<CarDriver>();
}
```

- Add the `[NotMapped]` `MakeName` property and the override for `ToString()`, both of which use the `MakeNavigation` reference navigation property:

```
public class Car : BaseEntity
{
  //omitted for brevity

  [NotMapped]
  public string MakeName => MakeNavigation?.Name ?? "Unknown";

  public override string ToString()
  {
    // Since the PetName column could be empty, supply
    // the default name of **No Name**.
    return $"{PetName ?? "**No Name**"} is a {Color} {MakeNavigation?.Name} with ID {Id}.";
  }
}
```

Step 2: Update the CarDriver Entity

- Update the `CarDriver` entity by adding the following reference navigation properties:

```
public class CarDriver : BaseEntity
{
  public int DriverId { get; set; }
  [ForeignKey(nameof(DriverId))]
  public Driver DriverNavigation { get; set; }
  [Column("InventoryId")]
  public int CarId { get; set; }
  [ForeignKey(nameof(CarId))]
  [InverseProperty(nameof(Car.CarDrivers))]
  public Car CarNavigation { get; set; }
}
```

Step 3: Update the Driver Entity

- Update the `Driver` entity by adding the following collection navigation properties:

```
public class Driver : BaseEntity
{
  public Person PersonInformation { get; set; } = new Person();
  [InverseProperty(nameof(Car.Drivers))]
  public ICollection<Car> Cars { get; set; } = new List<Car>();
  [InverseProperty(nameof(CarDriver.DriverNavigation))]
  public ICollection<CarDriver> CarDrivers { get; set; } = new List<CarDriver>();
}
```

Step 4: Update the Make Entity

- Update the `Make` entity by adding the following collection navigation property:

```
public class Make : BaseEntity
{
  //omitted for brevity
  [InverseProperty(nameof(Car.MakeNavigation))]
  public ICollection<Car> Cars { get; set; } = new List<Car>();
}
```

Step 5: Update the Radio Entity

- Update the `Radio` entity by adding the following reference and collection navigation properties:

```
public class Radio : BaseEntity
{
  //omitted for brevity
  [ForeignKey(nameof(CarId))]
  [InverseProperty(nameof(Car.RadioNavigation))]
  public Car CarNavigation { get; set; }
}
```

- The project should now build successfully.

Part 4: Create the Configuration Classes

The Fluent API is used to further define the models. The configuration classes will be used in the `ApplicationDbContext` in a later lab.

Copilot Agent Mode

Copilot creates the folder and the files, then adds the common configuration. The rest will be done manually.

```
Setup Prompt: Always use file scoped namespaces. Always combine attributes on a single line when possible. The project does not use nullable reference types. There is a GlobalUsings.cs file that includes common usings, don't include using statements in new files if they are already in the globalusings.cs file. I prefer expression bodied members when possible. Single line if statements should still use braces. Use ternary operators when appropriate. Use internal over private. All classes and methods are public unless told otherwise. Don't add a constructor unless instructed to do so. Use primary constructors when possible and don't declare a class level variable if the parameter from the constructor can be used. Don't initialize properties unless instructed to do so.

Prompt: Create a new folder named Configuration under the Entities folder in the Autolot.Models project.
In the folder, create a public class for each entity
(CarConfiguration.cs, MakeConfiguration.cs, RadioConfiguration.cs, DriverConfiguration.cs, CarDriverConfiguration.cs)
that implements IEntityTypeConfiguration<T> where T is the entity type.
For each configuration class, implement the Configure method. Do not add any configutation already handled by data annotations.
Add an entry in each class in the Configure method for the TimeStamp property to set it as IsRowVersion and to add a conversion to byte[].
```

```
Prompt: Update the CarConfiguration.cs class as follows:
Add class level two constants:
public const string IsNewQueryFilterName = "IsNew";
public const string IsDriveableFilterName = "IsDriveable";
Add the following to the configure method:
builder.HasQueryFilter(IsDriveableFilterName,c => c.IsDrivable);
builder.HasQueryFilter(IsNewQueryFilterName,c => c.DateBuilt > new DateTime(2020,1,1));
builder.Property(p => p.IsDrivable).HasDefaultValue(true);
builder.Property(e => e.DateBuilt).HasDefaultValueSql("getdate()");
builder.Property(e => e.Display)
    .HasComputedColumnSql("[PetName] + ' (' + [Color] + ')'", stored: true);
CultureInfo provider = new("en-us");
NumberStyles style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
builder.Property(p => p.Price).HasConversion(
        v => decimal.Parse(v, style, provider),
        v => v.ToString("C2"));
builder.HasOne(d => d.MakeNavigation).WithMany(p => p.Cars).HasForeignKey(d => d.MakeId)
        .OnDelete(DeleteBehavior.ClientSetNull).HasConstraintName("FK_Inventory_Makes_MakeId");
builder.HasMany(p => p.Drivers).WithMany(p => p.Cars).UsingEntity<CarDriver>(
    j => j.HasOne(cd => cd.DriverNavigation).WithMany(d => d.CarDrivers)
            .HasForeignKey(nameof(CarDriver.DriverId))
            .HasConstraintName("FK_InventoryDriver_Drivers_DriverId")
            .OnDelete(DeleteBehavior.Cascade),
    j => j.HasOne(cd => cd.CarNavigation).WithMany(c => c.CarDrivers)
            .HasForeignKey(nameof(CarDriver.CarId))
            .HasConstraintName("FK_InventoryDriver_Inventory_InventoryId")
            .OnDelete(DeleteBehavior.ClientCascade),
    j => {
            j.HasKey(x => x.Id);
            j.HasIndex(cd => new { cd.CarId, cd.DriverId }).IsUnique(true);
        });

Prompt: Update the CarDriverConfiguration class Configure method by adding the following:
builder.HasIndex(e => new { e.DriverId, e.CarId })
    .IsUnique()
    .HasDatabaseName("IX_InventoryToDrivers_DriverId_CarId");
builder.HasOne(e => e.DriverNavigation)
    .WithMany(d => d.CarDrivers)
    .HasForeignKey(e => e.DriverId);
builder.HasOne(e => e.CarNavigation)
    .WithMany(c => c.CarDrivers)
    .HasForeignKey(e => e.CarId);
builder.HasQueryFilter(CarConfiguration.IsDriveableFilterName, c => c.CarNavigation.IsDrivable);
builder.HasQueryFilter(CarConfiguration.IsNewQueryFilterName, c => c.CarNavigation.DateBuilt > new
DateTime(2020, 1, 1));
```

```
Prompt: Update the DriverConfiguration class Configure method by adding the following:
builder.ComplexProperty(cp => cp.PersonInformation,
    pd =>
    {
        pd.Property<string>(nameof(Person.FirstName))
            .HasColumnName(nameof(Person.FirstName))
            .HasColumnType("nvarchar(50)");
        pd.Property<string>(nameof(Person.LastName))
            .HasColumnName(nameof(Person.LastName))
            .HasColumnType("nvarchar(50)");
        pd.Property(p => p.FullName)
            .HasColumnName(nameof(Person.FullName))
            .HasComputedColumnSql("[LastName] + ', ' + [FirstName]");
        pd.IsRequired(true);
    });

Prompt: Update the RadioConfiguration class Configure method by adding the following:
builder.HasQueryFilter(e => e.CarNavigation.IsDrivable);
builder.HasOne(d => d.CarNavigation)
    .WithOne(p => p.RadioNavigation)
    .HasForeignKey<Radio>(d => d.CarId);

Prompt: Create a SeriLogEntryConfiguration.cs class in the same directory that implements
IEntityTypeConfiguration<T> where T is the entity type.
Implement the Configure method. Do not add any configutation already handled by data annotations.
Do not add the TimeStamp conversion.
Add the following to the Configure method:
builder.Property(e => e.Properties).HasColumnType("Xml");
        builder.Property(e => e.TimeStamp).HasDefaultValueSql("GetDate()");
        builder.Property(p => p.LineNumber).HasDefaultValue(0).HasSentinel(-1);

Prompt: Add the following global usings to the GlobalUsings.cs file if they do not already exist
(sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Models.Entities.Configuration;

Prompt: Create a new folder named Configuration under the ViewModels folder in the Autolot.Models
project.
In the folder, create a class CarViewModelConfiguration.cs class that implements
IEntityTypeConfiguration<CarViewModel>.
Implement the Configure method. Do not add any configutation already handled by data annotations.
Do not add the TimeStamp conversion.
Add the following to the Configure method:
builder.ToTable(t => t.ExcludeFromMigrations());
CultureInfo provider = new("en-us");
NumberStyles style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
builder.Property(p => p.Price)
    .HasConversion(
        v => decimal.Parse(v, style, provider),
        v => v.ToString("C2"));

Prompt: Add the following global usings to the GlobalUsings.cs file if they do not already exist
(sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Models.ViewModels.Configuration;
```

Please proceed to Part 5 after verifying the generated code.

Manual

- Start by creating a folder named `Configuration` under the Entities folder and a folder named `Configuration` in the `ViewModels` folder, and add the following to the `GlobalUsings.cs` file:

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

Step 1: Create the CarConfiguration

- Add a new class named `CarConfiguration` in the `Configuration` folder. Clear out the code and update it to match the following:

```
namespace AutoLot.Models.Entities.Configuration;
public class CarConfiguration : IEntityTypeConfiguration<Car>
{
  public const string IsNewQueryFilterName = "IsNew";
  public const string IsDriveableFilterName = "IsDriveable";
  public void Configure(EntityTypeBuilder<Car> builder)
  {
    builder.Property(e => e.TimeStamp).IsRowVersion().HasConversion<byte[]>();
    builder.HasQueryFilter(IsDriveableFilterName,c => c.IsDrivable);
    builder.HasQueryFilter(IsNewQueryFilterName,c => c.DateBuilt > new DateTime(2020,1,1));
    builder.Property(e => e.DateBuilt).HasDefaultValueSql("getdate()");
    builder.Property(e => e.Display)
           .HasComputedColumnSql("[PetName] + ' (' + [Color] + ')'", stored: true);
    CultureInfo provider = new("en-us");
    NumberStyles style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
    builder.Property(p => p.Price).HasConversion(
            v => decimal.Parse(v, style, provider),
            v => v.ToString("C2"));
    builder.HasOne(d => d.MakeNavigation).WithMany(p => p.Cars).HasForeignKey(d => d.MakeId)
           .OnDelete(DeleteBehavior.ClientSetNull).HasConstraintName("FK_Inventory_Makes_MakeId");
    builder.HasMany(p => p.Drivers).WithMany(p => p.Cars).UsingEntity<CarDriver>(
        j => j.HasOne(cd => cd.DriverNavigation).WithMany(d => d.CarDrivers)
          .HasForeignKey(nameof(CarDriver.DriverId))
          .HasConstraintName("FK_InventoryDriver_Drivers_DriverId")
          .OnDelete(DeleteBehavior.Cascade),
        j => j.HasOne(cd => cd.CarNavigation).WithMany(c => c.CarDrivers)
          .HasForeignKey(nameof(CarDriver.CarId))
          .HasConstraintName("FK_InventoryDriver_Inventory_InventoryId")
          .OnDelete(DeleteBehavior.ClientCascade),
        j => { j.HasKey(x => x.Id);
               j.HasIndex(cd => new { cd.CarId, cd.DriverId }).IsUnique(true);
             });
    }
}
```

Step 2: Create the CarDriverConfiguration

- Add a new class named `CarDriverConfiguration` in the `Configuration` folder and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;
public class CarDriverConfiguration : IEntityTypeConfiguration<CarDriver>
{
  public void Configure(EntityTypeBuilder<CarDriver> builder)
  {
    builder.HasIndex(e => new { e.DriverId, e.CarId })
      .IsUnique().HasDatabaseName("IX_InventoryToDrivers_DriverId_CarId");
     builder.Property(e => e.TimeStamp).IsRowVersion().HasConversion<byte[]>();
    builder.HasOne(e => e.DriverNavigation)
      .WithMany(d => d.CarDrivers)
      .HasForeignKey(e => e.DriverId);
    builder.HasOne(e => e.CarNavigation)
      .WithMany(c => c.CarDrivers)
      .HasForeignKey(e => e.CarId);
    builder.HasQueryFilter(CarConfiguration.IsDriveableFilterName,
       c => c.CarNavigation.IsDrivable);
    builder.HasQueryFilter(CarConfiguration.IsNewQueryFilterName,
      c => c.CarNavigation.DateBuilt > new DateTime(2020, 1, 1));
  }
}
```

Step 3: Create the DriverConfiguration

- Add a new class named `DriverConfiguration` in the `Configuration` folder and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;
public class DriverConfiguration : IEntityTypeConfiguration<Driver>
{
  public void Configure(EntityTypeBuilder<Driver> builder)
  {
    builder.Property(e => e.TimeStamp).IsRowVersion().HasConversion<byte[]>();
    builder.ComplexProperty(cp => cp.PersonInformation,
      pd =>
      {
        pd.Property<string>(nameof(Person.FirstName))
          .HasColumnName(nameof(Person.FirstName))
          .HasColumnType("nvarchar(50)");
        pd.Property<string>(nameof(Person.LastName))
          .HasColumnName(nameof(Person.LastName))
          .HasColumnType("nvarchar(50)");
        pd.Property(p => p.FullName)
          .HasColumnName(nameof(Person.FullName))
          .HasComputedColumnSql("[LastName] + ', ' + [FirstName]");
        pd.IsRequired(true);
    });
  }
}
```

Step 4: Create the MakeConfiguration

- Add a new class named `MakeConfiguration` in the `Configuration` folder and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;
public class MakeConfiguration : IEntityTypeConfiguration<Make>
{
  public void Configure(EntityTypeBuilder<Make> builder)
  {
    builder.Property(e => e.TimeStamp).IsRowVersion().HasConversion<byte[]>();
  }
}
```

Step 5: Create the RadioConfiguration

- Add a new class named `RadioConfiguration` in the `Configuration` folder and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;
public class RadioConfiguration : IEntityTypeConfiguration<Radio>
{
  public void Configure(EntityTypeBuilder<Radio> builder)
  {
    builder.Property(e => e.TimeStamp).IsRowVersion().HasConversion<byte[]>();
    builder.HasQueryFilter(e => e.CarNavigation.IsDrivable);
    builder.HasOne(d => d.CarNavigation).WithOne(p => p.RadioNavigation)
           .HasForeignKey<Radio>(d => d.CarId);
  }
}
```

Step 6: Create the SeriLogEntryConfiguration

- Add a new class named `SeriLogEntryConfiguration` in the `Configuration` folder and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;
public class SeriLogEntryConfiguration : IEntityTypeConfiguration<SeriLogEntry>
{
  public void Configure(EntityTypeBuilder<SeriLogEntry> builder)
  {
    builder.Property(e => e.Properties).HasColumnType("Xml");
    builder.Property(e => e.TimeStamp).HasDefaultValueSql("GetDate()");
    builder.Property(p => p.LineNumber).HasDefaultValue(0).HasSentinel(-1);
  }
}
```

Step 7: Create the CarViewModelConfiguration

- Add another folder named `Configuration` under the `ViewModels` folder. Add a new class named `CarViewModelConfiguration.cs` into the `Configuration` folder. Update the code for the class to the following:

```
namespace AutoLot.Models.ViewModels.Configuration;
public class CarViewModelConfiguration : IEntityTypeConfiguration<CarViewModel>
{
  public void Configure(EntityTypeBuilder<CarViewModel> builder)
  {
    builder.HasNoKey();
    builder.ToTable(t => t.ExcludeFromMigrations());
    CultureInfo provider = new("en-us");
    NumberStyles style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
    builder.Property(p => p.Price)
           .HasConversion(
                v => decimal.Parse(v, style, provider),
                v => v.ToString("C2"));
    }
}
```

Part 5: Add the Configuration to the Entities

Copilot Agent Mode

```
Prompt: Update each class in the Entities folder to include the EntityTypeConfiguration attribute
with the correct configuration file based on the type.

Prompt: Update CarViewModel.cs to include the EntityTypeConfiguration attribute with the correct
configuration file based on the type.
```

Manual

- Add the `[EntityTypeConfiguration]` attribute to the `Car` class (keeping the other attributes in place):

```
[EntityTypeConfiguration(typeof(CarConfiguration))]
public class Car : BaseEntity
```

- Add the `[EntityTypeConfiguration]` attribute to the `CarDriver` class:

```
[EntityTypeConfiguration(typeof(CarDriverConfiguration))]
public class CarDriver : BaseEntity
```

- Add the `[EntityTypeConfiguration]` attribute to the `Driver` class:

```
[EntityTypeConfiguration(typeof(DriverConfiguration))]
public class Driver : BaseEntity
```

- Add the `[EntityTypeConfiguration]` attribute to the `Make` class:

```
[EntityTypeConfiguration(typeof(MakeConfiguration))]
public class Make : BaseEntity
```


- Add the `[EntityTypeConfiguration]` attribute to the `Radio` class:

```
[EntityTypeConfiguration(typeof(RadioConfiguration))]
public class Radio : BaseEntity
```

- Add the `[EntityTypeConfiguration]` attribute to the `SeriLogEntry` class:

```
[EntityTypeConfiguration(typeof(SeriLogEntryConfiguration))]
public class SeriLogEntry
```

- Add the `[EntityTypeConfiguration]` attribute to the `CarViewModel` class:

```
[EntityTypeConfiguration(typeof(CarViewModelConfiguration))]
public class CarViewModel
```

Summary

In this lab, you created the Models (Entities) and the `ViewModels` for the applications.

Next steps

In the next part of this tutorial series, you will create the `DbContext` and `DbContextFactory` and run your first migration.