.NET 9 App Dev Hands-On Lab

EF Lab 8 (Optional) – Temporal Tables

This lab involves updating all the tables to be system versioned or temporal. Before starting this lab, you must have completed EF Lab 6. For the updated tests later in this lab, you must have completed EF Lab 7.

Part 1: Update the Entity Configurations

The section updates the Fluent API configuration for each entity to convert it into a temporal table.

Copilot Agent Mode

- Complete Steps 1-5 using the following prompts, then proceed to Part 2.

```
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.

Prompt: In the AutoLot.Models project, update the Configuration files for Car, CarDriver, Driver,
Make, and Radio to include temporal table configuration.
Use ValidTo and ValidFrom as the period columns. Name the history tables dbo.[EntityName]Audit,
for example, dbo.DriversAudit.
Except for Car, name it InventoryAudit, and CarDriver, name it InventoryToDriversAudit.
```

Manual

Step 1: Update the Car Entity Configuration

- Update the `CarConfiguration.cs` class in the `Configuration` folder by adding the following:

```
public class CarConfiguration : IEntityTypeConfiguration<Car>
{
  public void Configure(EntityTypeBuilder<Car> builder)
  {
    //omitted for brevity
    builder.ToTable( b => b.IsTemporal(t =>
    {
      t.HasPeriodEnd("ValidTo");
      t.HasPeriodStart("ValidFrom");
      t.UseHistoryTable("InventoryAudit", "dbo");
    }));
  }
}
```

Step 2: Update the CarDriver Entity Configuration

- Update the `CarDriverConfiguration.cs` class by adding the following:

```
public class CarDriverConfiguration : IEntityTypeConfiguration<CarDriver>
{
  public void Configure(EntityTypeBuilder<CarDriver> builder)
  {
    //omitted for brevity
    builder.ToTable( b => b.IsTemporal(t =>
    {
      t.HasPeriodEnd("ValidTo");
      t.HasPeriodStart("ValidFrom");
      t.UseHistoryTable("InventoryToDriversAudit", "dbo");
    }));
  }
}
```

Step 3: Update the Driver Entity Configuration

- Update the `DriverConfiguration.cs` class by adding the following:

```
public class DriverConfiguration : IEntityTypeConfiguration<Driver>
{
  public void Configure(EntityTypeBuilder<Driver> builder)
  {
    //omitted for brevity
    builder.ToTable(b => b.IsTemporal(t =>
    {
      t.HasPeriodEnd("ValidTo");
      t.HasPeriodStart("ValidFrom");
      t.UseHistoryTable("DriversAudit", "dbo");
    }));
  }
}
```

Step 4: Update the Make Entity Configuration

- Update the `MakeConfiguration.cs` class by adding the following:

```
public class MakeConfiguration : IEntityTypeConfiguration<Make>
{
  public void Configure(EntityTypeBuilder<Make> builder)
  {
    //omitted for brevity
    builder.ToTable( b => b.IsTemporal(t =>
    {
      t.HasPeriodEnd("ValidTo");
      t.HasPeriodStart("ValidFrom");
      t.UseHistoryTable("MakesAudit", "dbo");
    }));
  }
}
```


Step 5: Update the Radio Entity Configuration

- Update the `RadioConfiguration.cs` class by adding the following:

```
public class RadioConfiguration : IEntityTypeConfiguration<Radio>
{
  public void Configure(EntityTypeBuilder<Radio> builder)
  {
    //omitted for brevity
    builder.ToTable( b => b.IsTemporal(t =>
    {
       t.HasPeriodEnd("ValidTo");
       t.HasPeriodStart("ValidFrom");
       t.UseHistoryTable("RadiosAudit", "dbo");
    }));
  }
}
```

Part 2: Create the Migration for Table Updates

MAKE SURE ALL FILES ARE SAVED

- Open a command prompt or Package Manager Console in the `AutoLot.Dal` directory. Create a new migration by running the following command:

```
dotnet ef migrations add Temporal -c AutoLot.Dal.EfStructures.ApplicationDbContext
```

- Update the database by executing the migration:

```
dotnet ef database update
```

- Update the script of all the migrations by running the following CLI command:

```
dotnet ef migrations script -o allmigrations.sql -i
```

Part 3: Create the TemporalViewModel in AutoLot.Models

This class is used when querying temporal tables and does not correspond to a database table.

Copilot Agent Mode

```
Prompt: In the AutoLot.Models project, add a new class to the ViewModels folder named
TemporalViewModel.cs that is generic, constrained to BaseEntity, new(), and contains the following
public properties:
Entity (T)
ValidFrom (datetime)
ValidTo (datetime)
```

Manual

- Add a new class named `TemporalViewModel.cs` into the `ViewModels` folder and update the code to the following:

```
namespace AutoLot.Models.ViewModels;
public class TemporalViewModel<T> where T: BaseEntity, new()
{
  public T Entity { get; set; }
  public DateTime ValidFrom { get; set; }
  public DateTime ValidTo { get; set; }
}
```

Part 4: Add the Temporal Repositories

Copilot Agent Mode

```
Prompt: In the AutoLot.Dal project, add a new interface named ITemporalTableBaseRepo.cs into the
Repos\Interfaces\Base directory. Have it inherit from IBaseRepo<T> with the same constraints. Add
the following methods:
  IQueryable<TemporalViewModel<T>> GetAllHistory();
  IQueryable<TemporalViewModel<T>> GetHistoryAsOf(DateTime dateTime);
  IQueryable<TemporalViewModel<T>> GetHistoryBetween(DateTime startDateTime, DateTime
endDateTime);
  IQueryable<TemporalViewModel<T>> GetHistoryContainedIn(DateTime startDateTime, DateTime
endDateTime);
  IQueryable<TemporalViewModel<T>> GetHistoryFromTo(DateTime startDateTime, DateTime endDateTime);

Prompt: In the AutoLot.Dal project, add a new class named TemporalTableBaseRepo.cs into the
Repos\Base directory. Have it inherit from BaseRepo<T> with the same constraints, and implement
ITemporalTableBaseRepo. Implement all of the interfaces methods using the ConvertToUtc helper. Add
an internal helper method named
ConvertToUtc that takes a date time and uses TimeXZoneInfo to convert the time passed in. Add
another internal method named ExecuteQuery that takes an IQueryable<T>, ordered by "ValidFrom",
and project that into a new instance if TemporalViewModel.

Prompt: In the AutoLot.Dal project, update the following Interfaces to inherit from
ITemporalTableBaseRepo instead of IBaseRepo:
  ICarRepo,  ICarDriverRepo, IDriverRepo, IMakeRepo, IRadioRepo

Prompt: In the AutoLot.Dal project, update the following Repos to inherit from
TemporalTableBaseRepo instead of BaseRepo:
  CarRepo, CarDriverRepo, DriverRepo, MakeRepo, RadioRepoPart 5: Temporal Table Initialization
```

Manual

Step 1: Create the Base Repository Interfaces

- Add a new interface named `ITemporalTableBaseRepo.cs` into the `Repos\Interfaces\Base` directory in the `AutoLot.Dal` project and update it to the following:

```
namespace AutoLot.Dal.Repos.Interfaces.Base;
public interface ITemporalTableBaseRepo<T> : IBaseRepo<T> where T : BaseEntity, new()
{
  IQueryable<TemporalViewModel<T>> GetAllHistory();
  IQueryable<TemporalViewModel<T>> GetHistoryAsOf(DateTime dateTime);
  IQueryable<TemporalViewModel<T>> GetHistoryBetween(
    DateTime startDateTime, DateTime endDateTime);
  IQueryable<TemporalViewModel<T>> GetHistoryContainedIn(
    DateTime startDateTime, DateTime endDateTime);
  IQueryable<TemporalViewModel<T>> GetHistoryFromTo(
    DateTime startDateTime, DateTime endDateTime);
}
```

Step 2: Create the Temporal Table Base Repository

- Add a new class to the `Repos/Base` folder named `TemporalTableBaseRepo.cs`, and update it to the following:

```
namespace AutoLot.Dal.Repos.Base;
public abstract class TemporalTableBaseRepo<T>
  : BaseRepo<T>, ITemporalTableBaseRepo<T> where T : BaseEntity, new()
{
//implementation goes here
}
```

- Add the two constructors supported by the `BaseViewRepo`:

```
protected TemporalTableBaseRepo(ApplicationDbContext context) : base(context) {}
protected TemporalTableBaseRepo(DbContextOptions<ApplicationDbContext> options)
  : base(options) { }
```

- Add an internal helper to convert the current time to UTC:

```
internal DateTime ConvertToUtc(DateTime dateTime)
  => TimeZoneInfo.ConvertTimeToUtc(dateTime, TimeZoneInfo.Local);
```

- Add an internal helper to execute one of the temporal queries:

```
internal IQueryable<TemporalViewModel<T>> ExecuteQuery(IQueryable<T> query)
  => query.OrderBy(e => EF.Property<DateTime>(e, "ValidFrom"))
          .Select(e => new TemporalViewModel<T>
          {
            Entity = e,
            ValidFrom = EF.Property<DateTime>(e, "ValidFrom"),
            ValidTo = EF.Property<DateTime>(e, "ValidTo")
          });
```

- The public methods execute the five temporal queries:

```
public IQueryable<TemporalViewModel<T>> GetAllHistory()
  => ExecuteQuery(Table.TemporalAll());

public IQueryable<TemporalViewModel<T>> GetHistoryAsOf(DateTime dateTime)
  => ExecuteQuery(Table.TemporalAsOf(ConvertToUtc(dateTime)));

public IQueryable<TemporalViewModel<T>> GetHistoryBetween(
    DateTime startDateTime, DateTime endDateTime)
  => ExecuteQuery(Table.TemporalBetween(ConvertToUtc(startDateTime), ConvertToUtc(endDateTime)));

public IQueryable<TemporalViewModel<T>> GetHistoryContainedIn(
    DateTime startDateTime, DateTime endDateTime)
  => ExecuteQuery(Table.TemporalContainedIn(ConvertToUtc(startDateTime),
ConvertToUtc(endDateTime)));

public IQueryable<TemporalViewModel<T>> GetHistoryFromTo(
    DateTime startDateTime, DateTime endDateTime)
  => ExecuteQuery(Table.TemporalFromTo(ConvertToUtc(startDateTime), ConvertToUtc(endDateTime)));
```

Step 3: Update the Interface Files to Implement the Temporal Interface

- Update each of the five main table interfaces to implement `ITemporalTableBaseRepo<T>`:

```
//ICarDriverRepo.cs
public interface ICarDriverRepo : ITemporalTableBaseRepo<CarDriver>
  //omitted for brevity

//ICarRepo.cs
public interface ICarRepo : ITemporalTableBaseRepo<Car>
  //omitted for brevity

//IDriverRepo.cs
public interface IDriverRepo : ITemporalTableBaseRepo<Driver>
  //omitted for brevity

//IMakeRepo.cs
public interface IMakeRepo : ITemporalTableBaseRepo<Make>
  //omitted for brevity

//IRadioRepo.cs
public interface IRadioRepo : ITemporalTableBaseRepo<Radio>
  //omitted for brevity
```

Step 4: Update the Repos to Inherit from Temporal Base Repo

- Update each of the five main table repos to inherit from `TemporalTableBaseRepo<T>`:

```
//CarDriverRepo.cs
public class CarDriverRepo : TemporalTableBaseRepo<CarDriver>, ICarDriverRepo
  //omitted for brevity

//CarRepo.cs
public class CarRepo : : TemporalTableBaseRepo<Car>, ICarRepo
  //omitted for brevity

//DriverRepo.cs
public class DriverRepo : TemporalTableBaseRepo<Driver>, IDriverRepo
  //omitted for brevity

//MakeRepo.cs
public class MakeRepo : TemporalTableBaseRepo<Make>, IMakeRepo
  //omitted for brevity

//RadioRepo.cs
public class RadioRepo : TemporalTableBaseRepo<Radio>, IRadioRepo
  //omitted for brevity
```

Part 5: Temporal Table Initialization

Our initialization code clears out the tables and reseeds them with test values. Temporal tables also need their audit tables cleared. To do that, they must switch to standard tables, clear the history, and then switch back to temporal.

Step 1: Update the Package Reference for Temporal Table Runtime Support

To programmatically determine the history table associated with a temporal table at runtime, you need to get the design-time model for the context. The `Microsoft.EntityFrameworkCore.Design` package is used to get the model, but it is trimmed at runtime. The following disables package trimming.

- Comment out the `IncludeAssets` tag in the `AutoLot.Dal.csproj` file:

```
<PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="[10.0.*,11.0)">
  <!--<IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>-->
  <PrivateAssets>all</PrivateAssets>
</PackageReference>
```

Step 2: Get the Design Time Model

To work with temporal tables in EF Core, you must create an instance of IModel that represents the design-time model.

- Add the following method to the `SampleDataInitializer.cs` file:

```
internal static IModel GetDesignTimeModel(ApplicationDbContext context)
{
  var serviceCollection = new ServiceCollection();
  serviceCollection.AddDbContextDesignTimeServices(context);
  var serviceProvider = serviceCollection.BuildServiceProvider();
  return serviceProvider.GetService<IModel>();
}
```

Step 3: Clear the History Table

Clearing the history table requires removing the system versioning from the table, clearing the data, and then adding the system versioning back.

- Add the following method to the `SampleDataInitializer.cs` file:

```
internal static void ClearHistoryTable(
  ApplicationDbContext context,
  IModel designTimeModel,
  (string SchemaName, string TableName, string EntityName) entityInfo)
{
  var alterTable = $"ALTER TABLE {entityInfo.SchemaName}.{entityInfo.TableName} ";
  var setVersioningOff = $"SET (SYSTEM_VERSIONING = OFF)";
  var strategy = context.Database.CreateExecutionStrategy();
  strategy.Execute(() =>
  {
    using var trans = context.Database.BeginTransaction();
    var designTimeEntity = designTimeModel.FindEntityType(entityInfo.EntityName);
    var historySchema = designTimeEntity.GetHistoryTableSchema();
    var historyTable = designTimeEntity.GetHistoryTableName();
    var setVersioningOn =
      $"SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE={historySchema}.{historyTable}))";
#pragma warning disable EF1002 // Risk of vulnerability to SQL injection.
    context.Database.ExecuteSqlRaw($"{alterTable}{setVersioningOff}");
    context.Database.ExecuteSqlRaw($"DELETE FROM {historySchema}.{historyTable}");
    context.Database.ExecuteSqlRaw($"{alterTable}{setVersioningOn}");
#pragma warning restore EF1002 // Risk of vulnerability to SQL injection.
    trans.Commit();
  });
}
```

Step 4: Update the ClearData Method

- Update the `ClearData` method to the following:

```
internal static void ClearData(ApplicationDbContext context)
{
  //omitted for brevity
  IModel designTimeModel = GetDesignTimeModel(context);
  foreach (var entityName in entities)
  {
    //omitted for brevity
    if (entity.IsTemporal())
    {
      ClearHistoryTable(context,designTimeModel,(schemaName,tableName,entityName));
    }
  }
}
```

Part 6: Integration Testing Temporal Tables

Step 1: Update the Package Reference for Temporal Table Runtime Support

- Comment out the `IncludeAssets` tag in the `AutoLot.Dal.Tests.csproj` file:

```
<PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="[10.0.*,11.0)">
  <!--<IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>-->
  <PrivateAssets>all</PrivateAssets>
</PackageReference>
```

Step 2: Update the Existing Tests

- Several tests must be updated due to the two new fields (`ValidFrom`, `ValidTo`). Update the following tests in the `CarTests.cs` file in the `IntegrationTests` folder of the `AutoLot.Dal.Tests` project:

```
[Fact]
public void ShouldNotGetTheLemonsUsingFromSql()
{
  var entity = Context.Model.FindEntityType($"{typeof(Car).FullName}");
  var tableName = entity.GetTableName();
  var schemaName = entity.GetSchema();
#pragma warning disable EF1002 // Risk of vulnerability to SQL injection.
  var query = Context.Cars.FromSqlRaw(
    $"Select *,ValidFrom,ValidTo from {schemaName}.{tableName}");
#pragma warning restore EF1002 // Risk of vulnerability to SQL injection.
  var qs = query.ToQueryString();
  OutputHelper.WriteLine($"Query: {qs}");
  var cars = query.ToList();
  Assert.Equal(9, cars.Count);
}
[Theory]
[InlineData(1, 1)]
[InlineData(2, 1)]
[InlineData(3, 1)]
[InlineData(4, 2)]
[InlineData(5, 3)]
[InlineData(6, 1)]
public void ShouldGetTheCarsByMakeUsingFromSql(int makeId, int expectedCount)
{
  var entity = Context.Model.FindEntityType($"{typeof(Car).FullName}");
  var tableName = entity.GetTableName();
  var schemaName = entity.GetSchema();
#pragma warning disable EF1002 // Risk of vulnerability to SQL injection.
  var cars = Context.Cars
    .FromSqlRaw($"Select *,ValidFrom,ValidTo from {schemaName}.{tableName}")
    .Where(x => x.MakeId == makeId).ToList();
#pragma warning restore EF1002 // Risk of vulnerability to SQL injection.
  Assert.Equal(expectedCount, cars.Count);
}
```

```
[Fact]
public void ShouldGetTheCarsUsingFromSqlWithIgnoreQueryFilters()
{
  var entity = Context.Model.FindEntityType($"{typeof(Car).FullName}");
  var tableName = entity.GetTableName();
  var schemaName = entity.GetSchema();
#pragma warning disable EF1002 // Risk of vulnerability to SQL injection.
  var query = Context.Cars.FromSqlRaw(
    $"Select *,ValidFrom,ValidTo from {schemaName}.{tableName}").IgnoreQueryFilters();
#pragma warning restore EF1002 // Risk of vulnerability to SQL injection.
  var qs = query.ToQueryString();
  OutputHelper.WriteLine($"Query: {qs}");
  var cars = query.ToList();
  Assert.Equal(10, cars.Count);
}
[Fact]
public void ShouldGetOneCarUsingInterpolation()
{
  var carId = 1;
  var query = Context.Cars
    .FromSqlInterpolated($"Select *,ValidFrom,ValidTo from dbo.Inventory where Id = {carId}")
    .Include(x => x.MakeNavigation);
  var qs = query.ToQueryString();
  OutputHelper.WriteLine($"Query: {qs}");
  var car = query.First();
  Assert.Equal("Black", car.Color);
  Assert.Equal("VW", car.MakeNavigation.Name);
}
```

Step 3: Add a New Test

- To test accessing temporal table data, add the following test to the `MakeTests.cs` class in the `AutoLot.Dal.Tests` project:

```
[Fact]
public void ShouldGetAllHistoryRows()
{
  var make = new Make { Name = "TestMake" };
  _repo.Add(make);
  Thread.Sleep(2000);
  make.Name = "Updated Name";
  _repo.Update(make);
  Thread.Sleep(2000);
  _repo.Delete(make);
  var list = _repo.GetAllHistory().Where(x => x.Entity.Id == make.Id).ToList();
  Assert.Equal(2, list.Count);
  Assert.Equal("TestMake", list[0].Entity.Name);
  Assert.Equal("Updated Name", list[1].Entity.Name);
  Assert.Equal(list[0].ValidTo, list[1].ValidFrom);
}
```

Summary

In this lab, you converted the tables to system-versioned tables.