.NET 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.

Step 1: Update the Car Entity Configuration

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

```
namespace AutoLot.Models.Entities.Configuration;
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");
    }));
  }
}
```

Step 2: Update the CarDriver Entity Configuration

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

```
namespace AutoLot.Models.Entities.Configuration;
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");
    }));
  }
}
```

Step 3: Update the Driver Entity Configuration

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

```
namespace AutoLot.Models.Entities.Configuration;
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");
    }));
  }
}
```

Step 4: Update the Make Entity Configuration

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

```
namespace AutoLot.Models.Entities.Configuration;
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");
    }));
  }
}
```

Step 5: Update the Radio Entity Configuration

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

```
namespace AutoLot.Models.Entities.Configuration;
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");
    }));
  }
}
```

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:

```
[Windows]
dotnet ef migrations add Temporal -c AutoLot.Dal.EfStructures.ApplicationDbContext
[Non-Windows]
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 relate to a table in the database.

- 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

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()
{
  IEnumerable<TemporalViewModel<T>> GetAllHistory();
  IEnumerable<TemporalViewModel<T>> GetHistoryAsOf(DateTime dateTime);
  IEnumerable<TemporalViewModel<T>> GetHistoryBetween(
    DateTime startDateTime, DateTime endDateTime);
  IEnumerable<TemporalViewModel<T>> GetHistoryContainedIn(
    DateTime startDateTime, DateTime endDateTime);
  IEnumerable<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 static DateTime ConvertToUtc(DateTime dateTime)
  => TimeZoneInfo.ConvertTimeToUtc(dateTime, TimeZoneInfo.Local);
```

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

```
internal static IEnumerable<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 IEnumerable<TemporalViewModel<T>> GetAllHistory()
  => ExecuteQuery(Table.TemporalAll());

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

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

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

public IEnumerable<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 interfaces to implement `ITemporalTableBaseRepo<T>`:

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

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

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

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

//RadioRepo.cs
public class IRadioRepo : 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 have to change to normal tables, have the history cleared, and then change 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, the `Microsoft.EntityFrameworkCore.Design` package can't be trimmed, which it is by default.

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

```
<PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="[8.0.*,9.0)">
  <!--<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 the `IModel`, representing 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 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)
{
  var entities = new[]
  {  typeof(CarDriver).FullName, typeof(Driver).FullName, typeof(Radio).FullName,
     typeof(Car).FullName, typeof(Make).FullName,
  };
  IModel designTimeModel = GetDesignTimeModel(context);
  foreach (var entityName in entities)
  {
    var entity = context.Model.FindEntityType(entityName);
    var tableName = entity.GetTableName();
    var schemaName = entity.GetSchema();
#pragma warning disable EF1002 // Risk of vulnerability to SQL injection.
    context.Database.ExecuteSqlRaw($"DELETE FROM {schemaName}.{tableName}");
    context.Database.ExecuteSqlRaw($"DBCC CHECKIDENT (\"{schemaName}.{tableName}\", RESEED, 1);");
#pragma warning restore EF1002 // Risk of vulnerability to SQL injection.
    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="[8.0.*,9.0)">
  <!--<IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>-->
  <PrivateAssets>all</PrivateAssets>
</PackageReference>
```

Step 2: 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. This completes the data access portion of the hands-on lab.