Walkthrough 8 - EF Core Database Migrations

Setup

This walkthrough will introduce database migrations to the movie tracker app.

1. Start SQL Server.
2. Open MovieTracker from the end of the previous walkthrough.
3. If the movie_tracker database exists, delete it in SQL Server Object Explorer (SSOE).

Database Migrations

1. Open the Package Manager Console (PMC) from the menu Tools / NuGet Package Manager / Package Manager Console.
2. In the PMC heading, ensure the Default project is **MovieTracker**.
3. Type the command **Add-Migration init**, press Enter.
4. A new folder named Migrations will be created. Inspect the YYYYMMDDHHMMSS_init.cs file to see that it contains the code to create the database schema and seed the database.
5. Also, inspect the MovieTrackerContextModelSnapshot.cs file to see that it contains similar code. Notice the data type for Genre and Title are nvarchar(max); this will create potentially inefficient SQL statements.
6. In the PMC, issue the command **Update-Database**.
7. In SSOE, if localhost\sqlexpress isn't there, click the Add SQL Server button, set Server Name to **localhost\sqlexpress**.
8. Explore the new database to see the expected table as well as a migration history table.
9. Run the site, it functions as before. Add a new movie.

Movie.cs

1. Limit the maximum length of title and add a new property to the Movie class.
2.
```
public class Movie
{
    [Key]
    public int Id { get; set; }

    [Required]
    [MaxLength(100)]
    public string Title { get; set; }

    [DataType(DataType.Date)]
    [Display(Name = "Date Seen")]
    public DateTime? DateSeen { get; set; }

    public string Genre { get; set; }

    [Range(1, 10)]
    public int? Rating { get; set; }

    [Display(Name = "Release Year")]
    public int? ReleaseYear { get; set; }
}
```
3. Save the file.

Database Migrations

1. In the PMC, type the command **Add-Migration release_year**, press Enter.
2. Issue the command **Update-Database**.
3. In SSOE, show the Movies table data and note that no data has been lost.

Genre.cs

1. In the Models folder, add a new class named **Genre**. Add the **using System.ComponentModel.DataAnnotations;** directive.
2.
```
public class Genre
{
    [Key]
    public int Id { get; set; }

    [MaxLength(25)]
    public string GenreDescription { get; set; }

    // Navigation property
    public List<Movie> Movies { get; set; }
}
```
3. Save the file.

Movie.cs

1. Update Movie to use Genre as a relation. Add the **using System.ComponentModel.DataAnnotations.Schema;** directive.
2.
```
public class Movie
{
```

```
    [Key]
    public int Id { get; set; }

    [Required]
    public string Title { get; set; }

    [DataType(DataType.Date)]
    [Display(Name = "Date Seen")]
    public DateTime? DateSeen { get; set; }

    ~~public string Genre { get; set; }~~
    // Navigation property
    [ForeignKey("Genre")]
    [Display(Name = "Genre")]
    public int? GenreId { get; set; }

    // Navigation property
    public Genre Genre { get; set; }

    [Range(1, 10)]
    public int? Rating { get; set; }

    [Display(Name = "Release Year")]
    public int? ReleaseYear { get; set; }
}
```

3. Save the file.
4. Changing from the string Genre to the int GenreId will necessitate the changing of many files.

MovieTrackerContext.cs

1. Update the data context to include the Genre class.
2.
```
...
    }

    public DbSet<MovieTracker.Models.Movie> Movie { get; set; }

    public DbSet<Genre> Genres { get; set; }
}
```

3. Add a seed statement for the Action genre.
4.
```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Genre>().HasData(
        new Genre { Id = 1, GenreDescription = "Action" },
    );

    modelBuilder.Entity<Movie>().HasData(
    ...
```

Generating Code with Excel

1. Open genres.txt, select all text (Ctrl+A) and copy (Ctrl+C).
2. Open Excel and start a Blank workbook.
3. Paste the genres into column B of Sheet1.
4. In cell A1, enter **1**, in A2, enter **2**.
5. Select cells A1 and A2. Double-click the fill handle (the box in the bottom-right of the selection) to fill the series of numbers.
6. Sheet should look approximately like this:

1	Action
2	Adventure
3	Animation
4	Biography
5	Comedy
6	Crime
7	Documentary
8	Drama
9	Family
10	Fantasy
11	Film Noir
12	History
13	Horror
14	Music
15	Musical
16	Mystery
17	Romance
18	Sci-Fi
19	Short Film

20	Sport
21	Superhero
22	Thriller
23	War
24	Western

7. Copy the **new Genre { Id = 1, GenreDescription = "Action" },** seed statement to cell C1.
8. Edit cell C1, changing it to a formula that references cells A1 and B1, **="new Genre { Id = "&A1&", GenreDescription = """"&B1&"""" },"**.
9. Double-click the fill handle to copy the formula to the remaining cells.
10. Select all of these new cells and click Ctrl+C.

MovieTrackerContext.cs

1. Paste the values into the HasData parameter. Remove the comma on the last entry.
2.
```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Genre>().HasData(
        new Genre { Id = 1, GenreDescription = "Action" },
        new Genre { Id = 2, GenreDescription = "Adventure" },
        new Genre { Id = 3, GenreDescription = "Animation" },
        new Genre { Id = 4, GenreDescription = "Biography" },
        new Genre { Id = 5, GenreDescription = "Comedy" },
        new Genre { Id = 6, GenreDescription = "Crime" },
        new Genre { Id = 7, GenreDescription = "Documentary" },
        new Genre { Id = 8, GenreDescription = "Drama" },
        new Genre { Id = 9, GenreDescription = "Family" },
        new Genre { Id = 10, GenreDescription = "Fantasy" },
        new Genre { Id = 11, GenreDescription = "Film Noir" },
        new Genre { Id = 12, GenreDescription = "History" },
        new Genre { Id = 13, GenreDescription = "Horror" },
        new Genre { Id = 14, GenreDescription = "Music" },
        new Genre { Id = 15, GenreDescription = "Musical" },
        new Genre { Id = 16, GenreDescription = "Mystery" },
        new Genre { Id = 17, GenreDescription = "Romance" },
        new Genre { Id = 18, GenreDescription = "Sci-Fi" },
        new Genre { Id = 19, GenreDescription = "Short Film" },
        new Genre { Id = 20, GenreDescription = "Sport" },
        new Genre { Id = 21, GenreDescription = "Superhero" },
        new Genre { Id = 22, GenreDescription = "Thriller" },
        new Genre { Id = 23, GenreDescription = "War" },
        new Genre { Id = 24, GenreDescription = "Western" }
    );

    modelBuilder.Entity<Movie>().HasData(
    ...
```

3. Update the movie seed statement to use the genre id.
4.
```
...
modelBuilder.Entity<Movie>().HasData(
    new Movie
    {
        Id = 1,
        Title = "Birds of Prey",
        DateSeen = DateTime.Now.AddDays(-50),
        GenreId = "Action"1,
        Rating = 6
    },
    new Movie
    {
        Id = 2,
        Title = "Palm Springs",
        DateSeen = DateTime.Now.AddDays(-25),
        Rating = 7
    },
    new Movie
    {
        Id = 3,
        Title = "Hamilton",
        GenreId = "Drama"8
    });
}
```

5. Save the file.

UnitTest1

1. Update the InMemory database to use GenreId.
2.
```
private MovieTrackerContext CreateContext(string databaseName)
{
    var options = new DbContextOptionsBuilder<MovieTrackerContext>().UseInMemoryDatabase(databaseName: databaseName).Options;
    var context = new MovieTrackerContext(options);

    context.Movie.AddRange(
        new Movie
        {
            Id = 1,
            Title = "Car Chases and Explosions",
            DateSeen = new DateTime(2021, 7, 1).Date,
            GenreId = "Action"1,
            Rating = 6
        },
        new Movie
```

```
        {
            Id = 2,
            Title = "Silly Misunderstandings",
            DateSeen = new DateTime(2021, 8, 15).Date,
            GenreId = "Comedy"5,
            Rating = 7
        },
        new Movie
        {
            Id = 3,
            Title = "Serious Discussions",
            DateSeen = new DateTime(2021, 9, 30).Date,
            GenreId = "Drama"8,
            Rating = 8
        }
    );

    context.SaveChanges();
    return context;
}
```

3. Update the tests to use GenreId.
4.
```
[Fact]
public async Task Index_NoInput_ReturnsMovies()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Index();

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<List<Movie>>(viewResult.Model);
    var movies = viewResult.Model as List<Movie>;
    // Check the number of movies and a portion of every record and all fields
    Assert.Equal(3, movies.Count);
    Assert.Equal(1, movies[0].Id);
    Assert.Equal("Silly Misunderstandings", movies[1].Title);
    Assert.Equal(new DateTime(2021, 9, 30).Date, movies[2].DateSeen);
    Assert.Equal("Action"1, movies[0].GenreId);
    Assert.Equal(7, movies[1].Rating);
}
```

5.
```
[Fact]
public async Task Details_MovieId_ReturnsMovie()
{
    // Arrange
    var context = CreateContext("Details");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Details(1);

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<Movie>(viewResult.Model);
    var movie = viewResult.Model as Movie;
    // Test all properties
    Assert.Equal(1, movie.Id);
    Assert.Equal("Car Chases and Explosions", movie.Title);
    Assert.Equal(new DateTime(2021, 7, 1).Date, movie.DateSeen);
    Assert.Equal("Action"1, movie.GenreId);
    Assert.Equal(6, movie.Rating);
}
```

6.
```
[Fact]
public async Task Create_Movie_ReturnsToIndex()
{
    // Arrange
    var context = CreateContext("Create");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Create(
        new Movie
        {
            Title = "Testing for Fun and Profit",
            DateSeen = DateTime.Now.Date,
            GenreId = "Drama"8,
            Rating = 9
        });

    // Assert
    Assert.IsType<RedirectToActionResult>(actionResult);
    var redirectToActionResult = actionResult as RedirectToActionResult;
    Assert.Equal("Index", redirectToActionResult.ActionName);
    // Verify count
    actionResult = await moviesController.Index();
    var viewResult = actionResult as ViewResult;
    var movies = viewResult.Model as List<Movie>;
    Assert.Equal(4, movies.Count);
}
```

7. Save the file.
8. From the Build menu, select Rebuild Solution to ensure no other remnants of the Genre string remain.
9. Run all tests, they should pass.

Database Migrations

1. In the PMC, type the command **Add-Migration genre**, press Enter.
2. Issue the command **Update-Database**.
3. The updated model necessitates updating the views.

MoviesController.cs

1. Right-click in the Index method and select Add View..., select Razor View, click Add.
2. Set Template to **List**.
3. Set Model class to **Movie (MovieTracker.Models)**, click Add. Click Yes to replace existing view.
4. Run the site, navigate to Movies and note that the Genre doesn't appear.
5. Update the query to include Genre.
6. ```
   public async Task<IActionResult> Index()
   {
       return View(await _context.Movie.Include(m => m.Genre).ToListAsync());
   }
   ```
7. Save the file, refresh the browser. The GenreId appears.

Index.cshtml

1. Update the table row to display GenreDescription.
2. ```
   ...
   <td>
       @Html.DisplayFor(modelItem => item.Genre.~~Id~~GenreDescription)
   </td>
   ...
   ```
3. Save the file, refresh the browser. The GenreDescription appears.

MoviesController.cs

1. Right-click in the Details method and select Add View..., select Razor View, click Add.
2. Set Template to **Details**.
3. Set Model class to **Movie (MovieTracker.Models)**, click Add. Click Yes to replace existing view.

Details.cshtml

1. Update the definition to display GenreDescription.
2. ```
   ...
   <dd class = "col-sm-10">
       @Html.DisplayFor(model => model.Genre.~~Id~~GenreDescription)
   </dd>
   ...
   ```
3. Save the file.

MoviesController.cs

1. Update the Details method to include the Genre.
2. ```
   public async Task<IActionResult> Details(int? id)
   {
       if (id == null)
       {
           return View("Error",
               new ErrorViewModel
               {
                   Description = "Movie id invalid."
               });
       }

       var movie = await _context.Movie
           .Include(m => m.Genre)
           .FirstOrDefaultAsync(m => m.Id == id);
       if (movie == null)
       {
           return View("Error",
               new ErrorViewModel
               {
                   RequestId = id.ToString(),
                   Description = $"Unable to find movie with id={id}."
               });
       }
   ```
3. Save the file, return to the browser and access the details of an movie that has a Genre.

MoviesController.cs

1. Right-click in the Get Create method and select Add View..., select Razor View, click Add.
2. Set Template to **Create**.
3. Set Model class to **Movie (MovieTracker.Models)**, click Add. Click Yes to replace existing view.

Create.cshtml

1. Notice in the view that GenreId has an asp-items tag helper with a ViewBag.
2. Add autofocus to Title.
3.
```
...
<div class="form-group">
    <label asp-for="Title" class="control-label"></label>
    <input asp-for="Title" class="form-control" autofocus />
    <span asp-validation-for="Title" class="text-danger"></span>
</div>
...
```
4. Save the file, return to the browser and click the Create New link. Notice that GenreId appears with an empty drop-down list; this isn't ideal. Click Back to List.

MoviesController.cs

1. Create a new method to setup a select list for Genre.
2.
```
private SelectList GenreSelectList()
{
    return new SelectList(_context.Genres, "Id", "GenreDescription");
}
```
3. In the Get Create method, set up the view bag to pass a list to the view.
4.
```
public IActionResult Create()
{
    ViewBag.GenreId = GenreSelectList();
    return View();
}
```
5. Update the binding properties of the Post Create method to reflect the changes to the Model.
6.
```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Title,DateSeen,GenreId,Rating,ReleaseYear")] Movie movie)
{
    if (ModelState.IsValid)
    ...
```
7. Save the file, create a new movie.
8. Right-click in the Get Edit method and select Add View..., select Razor View, click Add.
9. Set Template to **Edit**.
10. Set Model class to **Movie (MovieTracker.Models)**, click Add. Click Yes to replace existing view.

Edit.cshtml

1. Add autofocus to Title.
2.
```
...
<div class="form-group">
    <label> asp-for="Title" class="control-label"></label>
    <input asp-for="Title" class="form-control" autofocus />
    <span> asp-validation-for="Title" class="text-danger"></span>
</div>
...
```
3. Save the file.

MoviesController.cs

1. In the Get Edit method, set up a view bag to pass a list to the view.
2.
```
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return View("Error",
            new ErrorViewModel
            {
                Description = "Movie id invalid."
            });
    }

    var movie = await _context.Movie.FindAsync(id);
    if (movie == null)
    {
        return View("Error",
            new ErrorViewModel
            {
                RequestId = id.ToString(),
                Description = $"Unable to find movie with id={id}."
            });
    }
    ViewBag.GenreId = new GenreSelectList();
    return View(movie);
}
```
3. Update the binding properties of the Post Edit method to reflect the changes to the Model.

4.
```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit([Bind("Id,Title,DateSeen,GenreId,Rating,ReleaseYear")] Movie movie)
{
    if (ModelState.IsValid)
    ...
```
5. Save the file, edit a movie.

MoviesController.cs

1. Right-click in the Get Delete method and select Add View..., select Razor View, click Add.
2. Set Template to **Delete**.
3. Set Model class to **Movie (MovieTracker.Models)**, click Add. Click Yes to replace existing view.

Delete.cshtml

1. Update the definition to display the genre description.
2.
```
...
<dt> class = "col-sm-2">
    @Html.DisplayNameFor(model => model.Genre)
</dt>
<dd class = "col-sm-10">
    @Html.DisplayFor(model => model.Genre.~~Id~~GenreDescription)
</dd class>
...
```
3. Save the file.

MoviesController.cs

1. Update the Get Delete method to include the Genre.
2.
```
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return View("Error",
            new ErrorViewModel
            {
                Description = "Movie id invalid."
            });
    }

    var movie = await _context.Movie
        .Include(m => m.Genre)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (movie == null)
    {
        return View("Error",
            new ErrorViewModel
            {
                RequestId = id.ToString(),
                Description = $"Unable to find movie with id={id}."
            });
    }

    return View(movie);
}
```
3. Save the file. Delete a movie.
4. Run all tests to ensure they still pass.