EF Core 5 Many-to-Many Relationships

If you're a Microsoft developer using EF Core, you are probably really excited about the new feature in **EF Core 5 that makes creating many-to-many relationships** super simple! Many-to-many relationships in EF Core 5 work intuitively now, so if you have installed the **.NET 5 SDK** or **Visual Studio 2019 16.9 preview 1** you can test the new feature pretty quickly from a .NET Core Console Application targeting .NET 5. In this EF Core 5 tutorial, I will walk you through the standard sample application of building a many-to-many relationship between blog posts and tags in a SQLite database. I'll also be using the **migrations feature in EF Core**, because we'll be able to see from the initial migration that indeed the proper database tables are being created to support many-to-many relationships in EF Core 5. I'll also be using the new [top-level programs feature in C# 9](https://www.davidhayden.me/blog/top-level-programs-in-csharp-9) to alleviate some of the boilerplate code and nesting of the code.

## .NET Core Console Application Targeting .NET 5

The first thing you need to do is create a .NET Core console application targeting the new .NET 5 Framework. Name the console application anything you want, but make sure it targets .NET 5.

dotnet new console -f net5.0 -n Sample

I'll be using SQLite as the database in this tutorial and as mentioned I will be using the migrations feature in EF Core, so I need to reference two Nuget packages in the .NET Core Console Application: **Microsoft.EntityFrameworkCore.Sqlite** and **Microsoft.EntityFrameworkCore.Design**, both v 5.0.0.

cd Sample

dotnet add package Microsoft.EntityFrameworkCore.Sqlite -v 5.0.0

dotnet add package Microsoft.EntityFrameworkCore.Design -v 5.0.0

If you haven't already, you can open the application in Visual Studio 2019 (16.9 Preview 1 is the latest at this time) or your favorite editor, such as Visual Studio Code. If you want to open it in Visual Studio Code, simply invoke the Visual Studio Code CLI right from the terminal.

code .

## Creating the EF Core 5 Entities

Of course, we're using a code-first approach with EF Core 5, so we'll create a blog Post entity and a Tag entity that will serve to create the sample many-to-many relationship.

public class Post

{

public int PostId { get; set; }

[Required]

[StringLength(50)]

public string Title { get; set; }

public ICollection<Tag> Tags { get; set; }

}

public class Tag

{

public int TagId { get; set; }

[Required]

[StringLength(25)]

public string Name { get; set; }

public ICollection<Post> Posts { get; set; }

}

## Create Custom DbContext in EF Core 5

We also need to create a custom context class that inherits from DbContext in EF Core 5. Migrations will be responsible for creating the migration files to create the "blog.db" SQLite database.

public class BlogContext : DbContext

{

public DbSet<Post> Posts { get; set; }

public DbSet<Tag> Tags { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlite("Data Source=blog.db");

}

}

Save everything and build the project to make sure there aren't any build errors.

## Run Migrations in EF Core 5

Migrations has everything it needs to create the initial migrations file. Open a terminal at the project's root directory.

If you haven't already, you'll need to install the **EF Core Tools**.

dotnet tool install --global dotnet-ef

If you have installed them but it's been awhile since you've updated them, you may want to update the EF Core command line tools to the latest version.

dotnet tool update --global dotnet-ef

We essentially want the tools to be version 5.0.0 or later.

dotnet ef

Entity Framework Core .NET Command-line Tools 5.0.0

Assuming you have the correct version of the EF Core command line tools installed, run migrations on the project and update the database.

dotnet ef migrations add InitialCreate

dotnet ef database update

## Check the EF Core 5 Migrations File

All we need to do to verify the proper database tables were created in the SQLite database by EF Core 5 is to check the InitialCreate migration file. Optionally, we can also check the database itself. If you view the migration file, you will notice that 3 tables are being created: Posts, Tags, and PostTag.

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "Posts",

columns: table => new

{

PostId = table.Column<int>(type: "INTEGER", nullable: false)

.Annotation("Sqlite:Autoincrement", true),

Title = table.Column<string>(type: "TEXT", maxLength: 50, nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Posts", x => x.PostId);

});

migrationBuilder.CreateTable(

name: "Tags",

columns: table => new

{

TagId = table.Column<int>(type: "INTEGER", nullable: false)

.Annotation("Sqlite:Autoincrement", true),

Name = table.Column<string>(type: "TEXT", maxLength: 25, nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Tags", x => x.TagId);

});

migrationBuilder.CreateTable(

name: "PostTag",

columns: table => new

{

PostsPostId = table.Column<int>(type: "INTEGER", nullable: false),

TagsTagId = table.Column<int>(type: "INTEGER", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_PostTag", x => new { x.PostsPostId, x.TagsTagId });

table.ForeignKey(

name: "FK\_PostTag\_Posts\_PostsPostId",

column: x => x.PostsPostId,

principalTable: "Posts",

principalColumn: "PostId",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_PostTag\_Tags\_TagsTagId",

column: x => x.TagsTagId,

principalTable: "Tags",

principalColumn: "TagId",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "IX\_PostTag\_TagsTagId",

table: "PostTag",

column: "TagsTagId");

}

## Testing EF Core 5 Many-to-Many Relationships

We can now run a quick test to make sure EF Core 5 is indeed populating the necessary tables for the many-to-many relationships.

var context = new BlogContext();

var tag = new Tag { Name = "ef" };

context.Add(new Post

{

Title = "Many-to-Many Relationships in EFCore 5",

Tags = new List<Tag> { tag }

});

context.SaveChanges();

If you open up the SQLite database in your favorite database tool, you'll see that indeed the Posts, Tags, and PostTag tables have been properly populated with the new data.

## Wrap Up

If this is your first time using EF Core, then you'll be thinking that this is how you would expect to create many-to-many relationships between entities, and you would be right. But, this intuitive syntax is new in EF Core 5. I won't elaborate on how many-to-many relationships were done before by early version of EF Core, but it wasn't as intuitive.

I highly recommend you give this a shot yourself. Spin up your own .NET Core Console Application targeting the new .NET 5 framework and use EF Core 5 to create many-to-many relationships between two different entities. Instead of using a blog example with blog post and tag entities, try it using a membership example with user and group entities.