**✅ Folder Structure**

dotnetapp/

│

├── Models/

│ ├── Author.cs

│ ├── Book.cs

│ ├── Publisher.cs

│ ├── Category.cs

│ └── ApplicationDbContext.cs

│

├── Program.cs

└── dotnetapp.csproj

**1. Author.cs**

using System.Collections.Generic;

namespace dotnetapp.Models

{

public class Author

{

public int AuthorID { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public ICollection<Book> Books { get; set; }

}

}

**2. Publisher.cs**

using System.Collections.Generic;

namespace dotnetapp.Models

{

public class Publisher

{

public int PublisherID { get; set; }

public string Name { get; set; }

public ICollection<Book> Books { get; set; }

}

}

**3. Category.cs**

using System.Collections.Generic;

namespace dotnetapp.Models

{

public class Category

{

public int CategoryID { get; set; }

public string Name { get; set; }

public ICollection<Book> Books { get; set; }

}

}

**4. Book.cs**

namespace dotnetapp.Models

{

public class Book

{

public int BookID { get; set; }

public string Title { get; set; }

public int PublisherID { get; set; }

public Publisher? Publisher { get; set; }

public int AuthorID { get; set; }

public Author? Author { get; set; }

public int CategoryID { get; set; }

public Category? Category { get; set; }

}

}

**5. ApplicationDbContext.cs**

using Microsoft.EntityFrameworkCore;

namespace dotnetapp.Models

{

public class ApplicationDbContext : DbContext

{

private string connectionString =

"User ID=sa;password=examlyMssql@123; server=localhost;Database=appdb;trusted\_connection=false;Persist Security Info=False;Encrypt=False";

// Default constructor

public ApplicationDbContext()

{

}

// Constructor for unit testing / dependency injection

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Author> Authors { get; set; }

public DbSet<Book> Books { get; set; }

public DbSet<Publisher> Publishers { get; set; }

public DbSet<Category> Categories { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// Only configure if not already configured externally (e.g., in testing)

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlServer(connectionString);

}

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Author → Book (1-to-many)

modelBuilder.Entity<Book>()

.HasOne(b => b.Author)

.WithMany(a => a.Books)

.HasForeignKey(b => b.AuthorID);

// Publisher → Book (1-to-many)

modelBuilder.Entity<Book>()

.HasOne(b => b.Publisher)

.WithMany(p => p.Books)

.HasForeignKey(b => b.PublisherID);

// Category → Book (1-to-many)

modelBuilder.Entity<Book>()

.HasOne(b => b.Category)

.WithMany(c => c.Books)

.HasForeignKey(b => b.CategoryID);

}

}

}

**6. Program.cs**

using System;

using dotnetapp.Models;

class Program

{

static void Main(string[] args)

{

using (var context = new ApplicationDbContext())

{

Console.WriteLine("Creating database 'appdb'...");

context.Database.EnsureCreated();

Console.WriteLine("Database created successfully.");

}

}

}

**7. Migration and Database Creation Commands**

Open terminal inside the dotnetapp folder:

dotnet new tool-manifest

dotnet tool install --local dotnet-ef --version 6.0.6

dotnet dotnet-ef

Then create migration and update the database:

dotnet dotnet-ef migrations add "InitialSetup"

dotnet dotnet-ef database update

**8. Run and Verify**

dotnet restore

dotnet build

dotnet run

Output:

Creating database 'appdb'...

Database created successfully.

Then, connect to SQL Server (sqlcmd -U sa -P examlyMssql@123) and verify:

USE appdb;

GO

SELECT name FROM sys.tables;

GO

You should see:

Authors

Books

Publishers

Categories

\_\_EFMigrationsHistory