Objective: Implement a Library Management System using EF Core with both Code First and Database First approaches, including CRUD operations, relationships, migrations, and advanced queries.

# 1) Basic CRUD Operations

We start by defining entities Book and Author with one-to-many relationship.

using System.Collections.Generic;  
using System.ComponentModel.DataAnnotations;  
using System.ComponentModel.DataAnnotations.Schema;  
  
namespace LibraryEF.Models  
{  
 public class Author  
 {  
 [Key]  
 public int AuthorId { get; set; }  
 [Required, MaxLength(200)]  
 public string Name { get; set; } = string.Empty;  
 public string? Bio { get; set; }  
  
   
 public ICollection<Book> Books { get; set; } = new List<Book>();  
 }  
  
 public class Book  
 {  
 [Key]  
 public int BookId { get; set; }  
 [Required, MaxLength(300)]  
 public string Title { get; set; } = string.Empty;  
  
 [ForeignKey("Author")]  
 public int AuthorId { get; set; }  
 public Author Author { get; set; } = null!;  
 }  
}

## LibraryContext.cs

using Microsoft.EntityFrameworkCore;  
  
namespace LibraryEF.Models  
{  
 public class LibraryContext : DbContext  
 {  
 public LibraryContext(DbContextOptions<LibraryContext> options) : base(options) { }  
  
 public DbSet<Book> Books { get; set; }  
 public DbSet<Author> Authors { get; set; }  
 }  
}

## CRUD Operations (Code First)

using System;  
using System.Linq;  
using LibraryEF.Models;  
  
class CrudDemo  
{  
 public static void Run(LibraryContext db)  
 {  
   
 var author = new Author { Name = "Robert C. Martin", Bio = "Uncle Bob" };  
 db.Authors.Add(author);  
 db.SaveChanges();  
  
 var book = new Book { Title = "Clean Architecture", AuthorId = author.AuthorId };  
 db.Books.Add(book);  
 db.SaveChanges();

var books = db.Books.Include(b => b.Author).ToList();  
 foreach (var b in books)  
 Console.WriteLine($"{b.Title} by {b.Author.Name}");  
  
   
 book.Title = "Clean Architecture (2nd Edition)";  
 db.SaveChanges();  
  
   
 db.Books.Remove(book);  
 db.SaveChanges();  
 }  
}

# 2) Advanced Configurations and Migrations

Introduce Genre entity and many-to-many relation with Book using Fluent API.

namespace LibraryEF.Models  
{  
 public class Genre  
 {  
 public int GenreId { get; set; }  
 public string Name { get; set; } = string.Empty;  
  
 public ICollection<Book> Books { get; set; } = new List<Book>();  
 }  
}

protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
 base.OnModelCreating(modelBuilder);  
  
 modelBuilder.Entity<Book>()  
 .HasMany(b => b.Genres)  
 .WithMany(g => g.Books)  
 .UsingEntity(j => j.ToTable("BookGenres"));  
}

Run migrations:

dotnet ef migrations add InitLibrary  
dotnet ef database update

# 3)Reverse Engineering

Assume database already has Books, Authors, Genres. Reverse engineer using EF Core CLI:

dotnet ef dbcontext scaffold "Server=.;Database=LibraryDb;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -o Models

# 4) EF Core Advanced Queries

using(var db = new LibraryContext(...))  
{  
 var books = db.Books  
 .Include(b => b.Author)  
 .Include(b => b.Genres)  
 .Where(b => b.Genres.Any(g => g.Name == "Programming"))  
 .ToList();  
  
 foreach(var b in books)  
 {  
 Console.WriteLine($"{b.Title} by {b.Author.Name} | Genres: {string.Join(",", b.Genres.Select(g=>g.Name))}");  
 }  
}

# 5) Sample Output

Clean Architecture by Robert C. Martin | Genres: Programming, Software Engineering  
Domain-Driven Design by Eric Evans | Genres: Programming, Architecture