upGrad ENTERPRISE

ASP.NET Core MVC with EF Core - Part-2

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

1. Relations in EF Core

Entity Framework Core (EF Core) allows us to model relationships between entities similar to database relationships.

Types of Relationships

- **One-to-Many**

  Example: A `Category` has many `Products`.

```
public class Category {
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Product> Products { get; set; }
}

public class Product {
    public int Id { get; set; }
    public string Name { get; set; }
    public int CategoryId { get; set; }
    public Category Category { get; set; }
}
```

- **One-to-One**

  Example: A `User` has one `Profile`.

```
public class User {
    public int Id { get; set; }
    public string Username { get; set; }
    public UserProfile Profile { get; set; }
}

public class UserProfile {
    public int Id { get; set; }
    public string Address { get; set; }
    public int UserId { get; set; }
    public User User { get; set; }
}
```

- **Many-to-Many**

  Example: A `Student` can enroll in many `Courses`, and a `Course` can have many `Students`.

```
public class Student {
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Course> Courses { get; set; }
}

public class Course {
    public int Id { get; set; }
    public string Title { get; set; }
    public ICollection<Student> Students { get; set; }
}
```

2. Async Operations in EF Core

Asynchronous methods help keep applications responsive and scalable.

- **ToListAsync()**
  Retrieves data asynchronously.

```
var products = await _context.Products.ToListAsync();
```

- **SaveChangesAsync()**
  Saves changes to the database asynchronously.

```
await _context.SaveChangesAsync();
```

Benefits:

- Non-blocking I/O
- Better performance in web applications
- Supports large scale apps with many concurrent users

3. LINQ with EF Core

LINQ (Language Integrated Query) is used to query the database with EF Core.

Common Queries

- **Filtering**

```
var expensiveProducts = await _context.Products
    .Where(p => p.Price > 1000)
    .ToListAsync();
```

- **Sorting**

```
var orderedProducts = await _context.Products
    .OrderBy(p => p.Name)
    .ToListAsync();
```

- Joining

```
var query = from p in _context.Products
            join c in _context.Categories
            on p.CategoryId equals c.Id
            select new { p.Name, c.Name };
```

- Aggregation

```
var totalProducts = await _context.Products.CountAsync();
```

4. Lazy Loading vs Eager Loading

When working with related entities:

- **Eager Loading**

  Loads related data immediately using `Include()`.

```
var products = await _context.Products
    .Include(p => p.Category)
    .ToListAsync();
```

- **Lazy Loading**
  Loads related data only when it is accessed. Requires navigation properties to be virtual.

```
public class Product {
    public int Id { get; set; }
    public string Name { get; set; }
    public virtual Category Category { get; set; }
}
```

Comparison

- **Eager Loading** → Good for predictable queries, reduces multiple trips to DB.
- **Lazy Loading** → Good when related data is rarely needed, but can cause N+1 query issues.

Eager Loading vs Lazy Loading

- By default, Latest versions of EF Core does **not** enable lazy loading. You need to explicitly configure it.
- **Best practice**: In ASP.NET Core MVC, it's usually better to use eager loading (Include) or explicit loading instead of lazy loading, because lazy loading can cause N+1 query issues and performance problems in views.