FMX Academy

INFORMAÇÕES			
MÓDULO		**AULA**	**INSTRUTOR**
05	**ASP.NET Core Web API – Fundamentos**	**07 - EF Core: Relacionamentos N:N com Payload, Fluent API e Mapeamento com AutoMapper**	**Guilherme Paracatu**

1. Conectando Tudo e Mapeando Eficientemente

OBJETIVO: Aprender a modelar, configurar (com **Fluent API**) e consultar **relacionamentos Muitos-para-Muitos (N:N) com payload**. Introduzir e aplicar o **AutoMapper** para automatizar o mapeamento entre Entidades e DTOs, resolvendo **referência cíclica** e tornando o código mais limpo.

FERRAMENTAS: VS Code, Projeto LojaApi, PostgreSQL.

Bem-vindos de volta! Hoje, vamos mergulhar fundo em um dos tipos de relacionamento mais comuns e interessantes em bancos de dados: o **Muitos-para-Muitos (N:N)**, especialmente quando a 'ponte' entre as tabelas carrega informações extras. Além disso, vamos dominar a configuração explícita com a **Fluent API** e introduzir uma ferramenta essencial que vai revolucionar a forma como lidamos com DTOs: o **AutoMapper**.

1.1. O Relacionamento Muitos-para-Muitos (N:N) com Payload

- **N:N:** Um Pedido pode ter muitos Produtos, e um Produto pode estar em muitos Pedidos. Isso requer uma **tabela de junção**.
- **A Novidade (Payload):** E se, além de ligar um Pedido a um Produto, precisarmos guardar **quantos** daquele produto estão naquele pedido? Essa informação (quantidade) não pertence nem ao Pedido nem ao Produto, mas à **relação** entre eles. Chamamos isso de **payload** na tabela de junção.
- **Impacto no EF Core:** Quando a tabela de junção tem colunas além das chaves estrangeiras (como a quantidade), o EF Core **não pode mais mapear o N:N implicitamente**. Precisamos criar uma **entidade C# explícita** para representar a tabela de junção (PedidoProduto).
- **A Mudança nos Relacionamentos:** O N:N direto vira **dois relacionamentos Um-para-Muitos (1:N)** com a entidade de junção no meio:
  - Pedido 1 --- * --- PedidoProduto
  - Produto 1 --- * --- PedidoProduto

1.2. Configuração Explícita com Fluent API (O Poder do OnModelCreating)

Como agora temos uma entidade extra e relacionamentos 1:N no lugar do N:N implícito, a **Fluent API** no OnModelCreating se torna essencial para dizer ao EF Core exatamente como

tudo se conecta, especialmente se os nomes no banco não seguem as convenções do EF Core.

- **O Que é o OnModelCreating?** É um método no DbContext onde usamos código C# (Fluent API) para configurar detalhadamente o mapeamento, sobrescrevendo as convenções do EF Core.

- **O Que Configurar para N:N com Payload:**
  1. Mapear a entidade de junção (PedidoProduto) para sua tabela (TB_PEDIDOS_PRODUTOS).
  2. Definir a Chave Primária Composta (id_pedido, id_produto).
  3. Configurar os dois relacionamentos 1:N (Pedido -> PedidoProduto e Produto -> PedidoProduto), especificando as chaves estrangeiras.

Analogia do "Ajuste Fino com o Arquiteto":

> Pense no OnModelCreating como a reunião que você faz com o arquiteto (ModelBuilder) para dar instruções específicas sobre a planta da casa: "Para a junção entre Pedidos e Produtos, use a tabela TB_PEDIDOS_PRODUTOS e as colunas id_pedido e id_produto como chaves".

1.3. O Problema da Repetição e a Solução com AutoMapper

Com essa estrutura mais complexa (Pedido -> PedidoProduto -> Produto), mapear manualmente para DTOs seria ainda mais complicado e repetitivo. É aqui que o **AutoMapper** brilha!

- **AutoMapper:** Biblioteca que automatiza a cópia de dados entre objetos (Entidade <-> DTO).

- **Como Funciona:**
  1. **Configuração (Profiles):** Define as regras de mapeamento uma vez (CreateMap<Origem, Destino>();).
  2. **Injeção (IMapper):** Injeta a interface IMapper onde precisar mapear.
  3. **Mapeamento (_mapper.Map):** Chama _mapper.Map<TipoDestino>(objetoOrigem).

2. Mão na Massa - Implementando N:N com Payload e AutoMapper

Passo 1: O Cenário - Banco de Dados com N:N e Payload

Garanta que seu banco LojaDb possui as tabelas TB_PEDIDOS, TB_PRODUTOS e a tabela de junção TB_PEDIDOS_PRODUTOS com a coluna quantidade.

```
-- Garanta que estas tabelas existem:
CREATE TABLE IF NOT EXISTS public."TB_CLIENTES" ( /* ... */ );
CREATE TABLE IF NOT EXISTS public."TB_CATEGORIAS" ( /* ... */ );
CREATE TABLE IF NOT EXISTS public."TB_PRODUTOS" ( /* ... */ );

CREATE TABLE IF NOT EXISTS public."TB_PEDIDOS" (
    id_pedido SERIAL PRIMARY KEY,
```

```
    data_pedido TIMESTAMPTZ NOT NULL,
    valor_total NUMERIC(10, 2) NOT NULL,
    id_cliente INT NOT NULL,
    CONSTRAINT "FK_TB_PEDIDOS_TB_CLIENTES"
        FOREIGN KEY (id_cliente)
        REFERENCES public."TB_CLIENTES" (id_cliente)
);

-- Tabela de Junção N:N com Payload
CREATE TABLE "TB_PEDIDOS_PRODUTOS" (
    "id_pedido" INT NOT NULL, -- Coluna FK para Pedidos
    "id_produto" INT NOT NULL, -- Coluna FK para Produtos
    "quantidade" INT NOT NULL, -- Coluna de Payload
    PRIMARY KEY ("id_pedido", "id_produto"),
    CONSTRAINT "fk_pedido" FOREIGN KEY ("id_pedido") REFERENCES
"TB_PEDIDOS"("id_pedido"),
    CONSTRAINT "fk_produto" FOREIGN KEY ("id_produto") REFERENCES
"TB_PRODUTOS"("id_produto")
);
```

Passo 2: Mapeamento Manual das Entidades (Incluindo a Entidade de Junção)

Entities/PedidoProduto.cs (Nova Entidade de Junção)

```
// Entities/PedidoProduto.cs
using System.ComponentModel.DataAnnotations.Schema;

namespace LojaApi.Entities
{
    // Esta classe representa a tabela de junção TB_PEDIDOS_PRODUTOS
    // Não precisa do [Table] aqui pois configuraremos na Fluent API
    public class PedidoProduto
    {
        // Chaves Estrangeiras que formam a Chave Primária Composta
        public int PedidoId { get; set; }
        public int ProdutoId { get; set; }

        // Payload
        public int Quantidade { get; set; }

        // Propriedades de Navegação
        public Pedido? Pedido { get; set; }
        public Produto? Produto { get; set; }
    }
}
```

Atualize Entities/Pedido.cs e Entities/Produto.cs

Adicione a coleção da entidade de junção (1:N).

```
// Em Entities/Pedido.cs
// Adicione:
public ICollection<PedidoProduto> PedidoProdutos { get; set; } = new
List<PedidoProduto>();

// Em Entities/Produto.cs
// Adicione:
public ICollection<PedidoProduto> PedidoProdutos { get; set; } = new
List<PedidoProduto>();
```

Passo 3: Criar os DTOs para o Novo Modelo

Precisamos de DTOs que representem os itens do pedido com a quantidade.

DTOs/ProdutoPedidoDto.cs (Item do Pedido no DTO)

```
// DTOs/ProdutoPedidoDto.cs
namespace LojaApi.DTOs
{
    // Representa um item (produto + quantidade) dentro de um PedidoDetalhadoDto
    public class ProdutoPedidoDto
    {
        public int ProdutoId { get; set; }
        public string NomeProduto { get; set; } = string.Empty;
        public decimal PrecoUnitario { get; set; }
        public int Quantidade { get; set; }
    }
}
```

DTOs/PedidoDetalhadoDto.cs

```
// DTOs/PedidoDetalhadoDto.cs
namespace LojaApi.DTOs
{
    public class PedidoDetalhadoDto
    {
        public int Id { get; set; }
        public DateTime DataPedido { get; set; }
        public decimal ValorTotal { get; set; }
        public int ClienteId { get; set; }
        public string NomeCliente { get; set; } = string.Empty;

        // Contém a lista de itens com quantidade
        public List<ProdutoPedidoDto> Itens { get; set; } = new
```

```
List<ProdutoPedidoDto>();
    }
}
```

DTOs/CriarPedidoDto.cs

```
// DTOs/CriarPedidoDto.cs
using System.ComponentModel.DataAnnotations;
namespace LojaApi.DTOs
{
    public class CriarPedidoDto
    {
        [Required]
        public int ClienteId { get; set; }
        [Required]
        [MinLength(1)]
        public List<ItemPedidoParaCriarDto> Itens { get; set; } = new
List<ItemPedidoParaCriarDto>();
    }
```

DTOs/ItemPedidoParaCriarDto.cs

```
    public class ItemPedidoParaCriarDto
    {
        [Required]
        public int ProdutoId { get; set; }
        [Required]
        [Range(1, 100)]
        public int Quantidade { get; set; }
    }
}
```

Passo 4: Atualizar o DbContext com Fluent API Explícita

Configure a nova entidade de junção e seus relacionamentos no OnModelCreating, incluindo o mapeamento de nome da tabela Pedidos.

```
// Data/LojaContext.cs
using LojaApi.Entities;
using Microsoft.EntityFrameworkCore;

namespace LojaApi.Data
{
    public class LojaContext : DbContext
    {
        public LojaContext(DbContextOptions<LojaContext> options) : base(options) {}
```

```
        // DbSets existentes e o novo para a entidade de junção
        public DbSet<Cliente> Clientes { get; set; }
        public DbSet<Categoria> Categorias { get; set; }
        public DbSet<Produto> Produtos { get; set; }
        public DbSet<Endereco> Enderecos { get; set; }
        public DbSet<Pedido> Pedidos { get; set; }
        public DbSet<PedidoProduto> PedidoProdutos { get; set; } // DbSet para a
entidade de junção

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            // --- MAPEAMENTO DAS ENTIDADES (Fluent API) ---
            modelBuilder.Entity<Cliente>(entity => {
                entity.ToTable("TB_CLIENTES");
                entity.HasKey(c => c.Id);
                entity.Property(c => c.Id).HasColumnName("id_cliente");
                entity.Property(c =>
c.Nome).HasColumnName("nome_cliente").HasMaxLength(150).IsRequired();
                entity.Property(c =>
c.Email).HasColumnName("email_cliente").HasMaxLength(150).IsRequired();
                entity.HasIndex(c => c.Email).IsUnique();
                entity.Property(c => c.Ativo).HasColumnName("ativo");
                entity.Property(c => c.DataCadastro).HasColumnName("data_cadastro");
            });
            modelBuilder.Entity<Endereco>(entity => {
                entity.ToTable("TB_ENDERECOS");
                entity.HasKey(e => e.Id);
                entity.Property(e => e.Id).HasColumnName("id_cliente");
                // ... resto do mapeamento de Endereco ...
            });
            modelBuilder.Entity<Categoria>(entity => {
                entity.ToTable("TB_CATEGORIAS");
                entity.HasKey(c => c.Id);
                entity.Property(c => c.Id).HasColumnName("id_categoria");
                // ... resto do mapeamento de Categoria ...
            });
            modelBuilder.Entity<Produto>(entity => {
                entity.ToTable("TB_PRODUTOS");
                entity.HasKey(p => p.Id);
                entity.Property(p => p.Id).HasColumnName("id_produto");
                // ... resto do mapeamento de Produto ...
                entity.Property(p => p.CategoriaId).HasColumnName("id_categoria");
            });

            // **MAPEAMENTO DA TABELA PEDIDOS**
            modelBuilder.Entity<Pedido>(entity => {
                entity.ToTable("TB_PEDIDOS"); // Especifica o nome correto da tabela
                entity.HasKey(p => p.Id);
                entity.Property(p => p.Id).HasColumnName("id_pedido");
                entity.Property(p => p.DataPedido).HasColumnName("data_pedido");
                entity.Property(p =>
p.ValorTotal).HasColumnName("valor_total").HasColumnType("decimal(10, 2)");
                entity.Property(p => p.ClienteId).HasColumnName("id_cliente");
```

```
            });

            // --- MAPEAMENTO DA ENTIDADE DE JUNÇÃO ---
            modelBuilder.Entity<PedidoProduto>(entity =>
            {
                entity.ToTable("TB_PEDIDOS_PRODUTOS");
                // Define a Chave Primária Composta
                entity.HasKey(pp => new { pp.PedidoId, pp.ProdutoId });
                // Mapeia as colunas
                entity.Property(pp => pp.PedidoId).HasColumnName("id_pedido");
                entity.Property(pp => pp.ProdutoId).HasColumnName("id_produto");
                entity.Property(pp => pp.Quantidade).HasColumnName("quantidade");
            });

            // --- CONFIGURAÇÃO DOS RELACIONAMENTOS ---

            // Relacionamento 1:1 Cliente <-> Endereco
            modelBuilder.Entity<Cliente>()
                .HasOne(c => c.Endereco).WithOne(e =>
e.Cliente).HasForeignKey<Endereco>(e => e.Id);

            // Relacionamento 1:N Cliente -> Pedidos
            modelBuilder.Entity<Cliente>()
                .HasMany(c => c.Pedidos).WithOne(p => p.Cliente).HasForeignKey(p =>
p.ClienteId);

            // Relacionamento 1:N Categoria -> Produtos
            modelBuilder.Entity<Categoria>()
                .HasMany(c => c.Produtos).WithOne(p => p.Categoria).HasForeignKey(p =>
p.CategoriaId);

            // Relacionamento N:N é modelado como dois 1:N com a entidade de junção:
            // 1. Pedido -> PedidoProduto
            modelBuilder.Entity<Pedido>()
                .HasMany(p => p.PedidoProdutos) // Um Pedido tem muitos PedidoProdutos
                .WithOne(pp => pp.Pedido)      // Um PedidoProduto pertence a um Pedido
                .HasForeignKey(pp => pp.PedidoId) // A chave estrangeira está em
PedidoProduto.PedidoId
                .HasConstraintName("fk_pedido");

            // 2. Produto -> PedidoProduto
            modelBuilder.Entity<Produto>()
                .HasMany(p => p.PedidoProdutos) // Um Produto está em muitos
PedidoProdutos
                .WithOne(pp => pp.Produto)     // Um PedidoProduto refere-se a um
Produto
                .HasForeignKey(pp => pp.ProdutoId) // A chave estrangeira está em
PedidoProduto.ProdutoId
                .HasConstraintName("fk_produto");
        }
    }
}
```

Passo 5: Instalar e Configurar o AutoMapper

1. **Instalar Pacote:** dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection
2. **Criar o Perfil de Mapeamento:** Crie Mappings/MappingProfile.cs.

```
// Mappings/MappingProfile.cs
using AutoMapper;
using LojaApi.DTOs;
using LojaApi.Entities;

namespace LojaApi.Mappings
{
    public class MappingProfile : Profile
    {
        public MappingProfile()
        {
            // Mapeia a entidade de junção PedidoProduto para o DTO do item
            CreateMap<PedidoProduto, ProdutoPedidoDto>()
                .ForMember(dest => dest.ProdutoId, opt => opt.MapFrom(src =>
src.ProdutoId))
                .ForMember(dest => dest.NomeProduto, opt => opt.MapFrom(src =>
src.Produto != null ? src.Produto.Nome : string.Empty))
                .ForMember(dest => dest.PrecoUnitario, opt => opt.MapFrom(src =>
src.Produto != null ? src.Produto.Preco : 0m))
                .ForMember(dest => dest.Quantidade, opt => opt.MapFrom(src =>
src.Quantidade));

            // Mapeia a entidade Pedido para o DTO detalhado, incluindo a lista de
itens
            CreateMap<Pedido, PedidoDetalhadoDto>()
                .ForMember(dest => dest.NomeCliente, opt => opt.MapFrom(src =>
src.Cliente != null ? src.Cliente.Nome : string.Empty))
                .ForMember(dest => dest.Itens, opt => opt.MapFrom(src =>
src.PedidoProdutos)); // Mapeia a coleção da entidade de junção

            // Mapeamento de Produto (Entidade) para ProdutoResumoDto (DTO)
            CreateMap<Produto, ProdutoResumoDto>();
        }
    }
}
```

3. **Registrar o AutoMapper no Program.cs:**

```
// Program.cs
using LojaApi.Mappings;
builder.Services.AddAutoMapper(typeof(MappingProfile));
```

Passo 6: Refatorar os Serviços e Repositórios

Repositories/PedidoDBRepository.cs

```
// Repositories/PedidoDBRepository.cs
using LojaApi.Data; using LojaApi.Entities; using LojaApi.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;
namespace LojaApi.Repositories {
    public class PedidoDBRepository : IPedidoRepository {
        private readonly LojaContext _context;
        public PedidoDBRepository(LojaContext context) { _context = context; }

        public Pedido Adicionar(Pedido novoPedido) {
            _context.Pedidos.Add(novoPedido);
            _context.SaveChanges();
            return novoPedido;
        }

        public Pedido? ObterPorId(int id) {
            // Include em múltiplos níveis: Pedido -> Cliente e Pedido ->
PedidoProdutos -> Produto
            return _context.Pedidos
                .Include(p => p.Cliente) // Inclui o cliente
                .Include(p => p.PedidoProdutos) // Inclui a entidade de junção
                    .ThenInclude(pp => pp.Produto) // A partir da junção, inclui o
Produto associado
                .FirstOrDefault(p => p.Id == id);
        }

        public List<Pedido> ObterTodos() {
             return _context.Pedidos
                .Include(p => p.Cliente)
                .Include(p => p.PedidoProdutos).ThenInclude(pp => pp.Produto)
                .ToList();
        }
    }
}
```

Services/PedidoService.cs

```
// Services/PedidoService.cs
using AutoMapper;
using LojaApi.DTOs;
using LojaApi.Entities;
```

```
using LojaApi.Repositories.Interfaces;
using LojaApi.Services.Interfaces;

namespace LojaApi.Services {
    public class PedidoService : IPedidoService {
        private readonly IPedidoRepository _pedidoRepository;
        private readonly IClienteRepository _clienteRepository;
        private readonly IProdutoRepository _produtoRepository;
        private readonly IMapper _mapper;

        public PedidoService(IPedidoRepository pedidoRepository, IClienteRepository
clienteRepository, IProdutoRepository produtoRepository, IMapper mapper) {
            _pedidoRepository = pedidoRepository;
            _clienteRepository = clienteRepository;
            _produtoRepository = produtoRepository;
            _mapper = mapper;
        }

        public Pedido Adicionar(CriarPedidoDto pedidoDto) {
            var cliente = _clienteRepository.ObterPorId(pedidoDto.ClienteId);
            if (cliente == null) throw new Exception("Cliente não encontrado.");

            var pedidoProdutos = new List<PedidoProduto>();
            decimal valorTotalCalculado = 0;

            foreach (var itemDto in pedidoDto.Itens) {
                var produto = _produtoRepository.ObterPorId(itemDto.ProdutoId);
                if (produto == null) throw new Exception($"Produto com ID
{itemDto.ProdutoId} não encontrado.");
                if (produto.Estoque < itemDto.Quantidade) throw new Exception($"Estoque
insuficiente para {produto.Nome}.");

                // Mapeia o DTO do item para a entidade de junção usando AutoMapper
                var pedidoProdutoItem = _mapper.Map<PedidoProduto>(itemDto);
                pedidoProdutos.Add(pedidoProdutoItem);

                valorTotalCalculado += produto.Preco * itemDto.Quantidade;
                produto.Estoque -= itemDto.Quantidade; //Abate do estoque
(Simplificado)
            }

            var novoPedido = new Pedido {
                ClienteId = pedidoDto.ClienteId,
                DataPedido = DateTime.UtcNow,
                ValorTotal = valorTotalCalculado,
                PedidoProdutos = pedidoProdutos // Atribui a coleção da entidade de
junção
            };

            var pedidoAdicionado = _pedidoRepository.Adicionar(novoPedido);
            return _pedidoRepository.ObterPorId(pedidoAdicionado.Id) ??
pedidoAdicionado;
        }
```

```
        public PedidoDetalhadoDto? ObterDetalhesPorId(int id) {
            var pedido = _pedidoRepository.ObterPorId(id);
            if (pedido == null) return null;
            return _mapper.Map<PedidoDetalhadoDto>(pedido); // Mapeamento automático
        }

        // Implemente a interface IProdutoService e o método ObterTodosDetalhado
        // (Precisa do método ObterTodos no IPedidoRepository e na implementação)
        /*
        public List<PedidoDetalhadoDto> ObterTodosDetalhado()
        {
             var pedidos = _pedidoRepository.ObterTodos();
             return _mapper.Map<List<PedidoDetalhadoDto>>(pedidos);
        }
        */
    }
}
```

Passo 7: Configurar DI e Atualizar Controllers

Program.cs (Garantir que todos os repositórios estão registrados)

```
// Program.cs
builder.Services.AddScoped<IPedidoRepository, PedidoDBRepository>();
// ... outros registros ...
```

Controllers/PedidosController.cs

```
// Controllers/PedidosController.cs
using LojaApi.DTOs; using LojaApi.Services.Interfaces; using Microsoft.AspNetCore.Mvc;
namespace LojaApi.Controllers {
    [ApiController] [Route("api/[controller]")]
    public class PedidosController : ControllerBase {
        private readonly IPedidoService _pedidoService;
        public PedidosController(IPedidoService pedidoService) { _pedidoService =
pedidoService; }

        [HttpGet("{id}")]
        public ActionResult<PedidoDetalhadoDto> GetById(int id) { // Retorna DTO
            var pedidoDto = _pedidoService.ObterDetalhesPorId(id);
            if (pedidoDto == null) return NotFound();
            return Ok(pedidoDto);
        }
```

```
        [HttpPost]
        public ActionResult<PedidoDetalhadoDto> Add([FromBody] CriarPedidoDto
pedidoDto) { // Aceita DTO, Retorna DTO
            try {
                var pedidoCriado = _pedidoService.Adicionar(pedidoDto);
                var dtoRetorno = _pedidoService.ObterDetalhesPorId(pedidoCriado.Id);
                // Retorna o DTO detalhado para consistência
                return CreatedAtAction(nameof(GetById), new { id = pedidoCriado.Id },
dtoRetorno);
            }
            catch (Exception ex) { return BadRequest(ex.Message); }
        }
    }
}
```

Parte 4: Teste e Verificação

1. **Rode a API:** dotnet run.
2. **Acesse o Swagger** e teste os *Endpoints*.

O que testar:

- **POST /api/pedidos:** Crie um pedido com clienteId e uma lista de itens contendo produtoId e quantidade.

```
JSON
{
  "clienteId": 1,
  "itens": [
    { "produtoId": 1, "quantidade": 1 },
    { "produtoId": 3, "quantidade": 2 }
  ]
}
```

- **GET /api/pedidos/{id}:** Verifique se o pedido criado é retornado com os dados do cliente e a lista de Itens (contendo produtoId, nomeProduto, precoUnitario e quantidade).