02

# Alterando a Quantidade no Banco de Dados

### Transcrição

Chegamos no `ItemPedidoRepository`, e começaremos a implementar o método `UpdateQuantidade()` para gravarmos informações no banco de dados.

```
Public class ItemPedidoRepository : BaseRepository<ItemPedido>, IItemPedidoRepository
{
    public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
    {
    }

    public void UpdateQuantidade(ItemPedido itemPedido)
    {
        throw new NotImplementedException();
    }
}
```

Removeremos a linha `throw new NotImplementedException()` e começaremos a acessar a entidade `ItemPedido`. Vamos entender como faremos isso:

`ItemPedidoRepository` herda da classe `BaseRepository`, quando navegamos por essa classe - pasta seleciona-la e pressionar o atalho "F12" - encontraremos um campo privado chamado `dbSet` do tipo `Dbset<T>`, isto é, um tipo genérico que é o item pedido. No construtor da classe `dbSet` recebe o valor `contexto.Set<T>()`, método do Entity Framework que permite a manipulação da entidade em questão, no caso, `ItemPedido`.

```
namespace CasaDoCodigo.Repositories
{
    public abstract class BaseRepository<T> where T : BaseModel
    {
        protected readonly ApplicationContext contexto;
        protected readonly DbSet<T> dbSet;

        public BaseRepository(ApplicationContext contexto)
        {
            this.contexto = contexto;
            dbSet = contexto.Set<T>();

        }
    }
}
```

Voltaremos ao `ItemPedidoRepository` e acessaremos o `dbSet` no `UpdateQuantidade()`. Aplicaremos um filtro, de forma que localizemos somente o pedido que nos interessa, ou melhor, aquele que possui o `id` passado pelo objeto. Para aplicarmos esse filtro utilizaremos um método do Linq chamado `Where()`, que por sua vez receberá uma expressão lambda `ip` (que representa item pedido), `ip.id` que deverá ser igual ao `id` do parâmetro de `itemPedido`, portanto escreveremos `itemPedido.Id`.

```
public class ItemPedidoRepository: BaseRepository<ItemPedido>, IItemPedidoRepository
{
    public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
    {
    }

    public void UpdateQuantidade(ItemPedido itemPedido)
    {
        dbSet
            .Where(ip => ip.Id == itemPedido.Id)
```

Como estamos buscando apenas um item, usaremos outro método do Linq chamado `SingleOrDefault()` que retornará um elemento, caso esse elemento não seja encontrado será retornado um valor nulo.

```
public class ItemPedidoRepository: BaseRepository<ItemPedido>, IItemPedidoRepository
{
    public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
    {
    }

    public void UpdateQuantidade(ItemPedido itemPedido)
    {
        dbSet
            .Where(ip => ip.Id == itemPedido.Id)
            .SingleOrDefault();
```

Precisamos armazenar esse valor em algum lugar, portanto declararemos uma variável nova chamada `itemPedidoDB` .

```
public class ItemPedidoRepository: BaseRepository<ItemPedido>, IItemPedidoRepository
{
    public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
    {
    }

    public void UpdateQuantidade(ItemPedido itemPedido)
    {
        var itemPedidoDB =
        dbSet
            .Where(ip => ip.Id == itemPedido.Id)
            .SingleOrDefault();
```

Uma vez que estamos obtendo a variável `ItemPedidoDB` , precisamos verificar se o seu valor não é nulo, e caso seja, queremos ignorá-la. Adicionaremos uma verificação `If` , e caso o valor da variável seja diferente de nulo nós iremos atualizar o banco de dados.

Para atualizarmos os dados basta acessar o objeto `itemPedidoBD` e modificar sua quantidade, atribuindo a nova quantidade a ser recebida do parâmetro de `ItemPedido` .

```
public class ItemPedidoRepository: BaseRepository<ItemPedido>, IItemPedidoRepository
{
    public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
    {
```

```
    }

    public void UpdateQuantidade(ItemPedido itemPedido)
    {
        var itemPedidoDB =
        dbSet
            .Where(ip => ip.Id == itemPedido.Id)
            .SingleOrDefault();

        if (itemPedidoDB != null)
        {

            itemPedidoDB.Quantidade = itemPedido.Quantidade;
        }
```

No entanto, perceberemos que não podemos fazer atribuição a `Quantidade`, porque a propriedade `ItemPedido.Quantidade` não pode ser utilizada nesse contexto, porque o "acessador set é inacessível", segundo a mensagem no Visual Studio. Isso quer dizer que esta é apenas uma propriedade de leitura. Ao invés de modificarmos a propriedade para deixá-la pública, criaremos um novo método para modificar a `Quantidade`. Criaremos um novo método chamado `AtualizaQuantidade()`, que receberá como parâmetro a nova quantidade.

```
public class ItemPedidoRepository: BaseRepository<ItemPedido>, IItemPedidoRepository
{
    public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
    {
    }

    public void UpdateQuantidade(ItemPedido itemPedido)
    {
        var itemPedidoDB =
        dbSet
            .Where(ip => ip.Id == itemPedido.Id)
            .SingleOrDefault();

        if (itemPedidoDB != null)
        {
            itemPedidoDB.AtualizaQuantidade (itemPedido.Quantidade);
        }
```

Nós escrevemos o método, mas ele ainda não existe e por isso forçaremos o Visual Studio a criá-lo: selecionaremos o método e teclaremos o atalho "Ctrl + .", depois escolheremos a opção "Gerar método 'ItemPedido.AtualizaQuantidade'". Dessa forma não será mais indicado erro no código, mas não temos o método implementado. Selecionaremos `AtualizaQuantidade()` e pressionaremos "F12".

Implementaremos o método atribuindo à propriedade `Quantidade` o novo valor da `quantidade` recebida no método

```
namespace CasaDoCodigo.Models
{

<****!****>

 public ItemPedido(Pedido pedido, Produto produto, int quantidade, decimal precoUnitario)
        {
            Pedido = pedido;
```

```
            Produto = produto;
            Quantidade = quantidade;
            PrecoUnitario = precoUnitario;
        }


        internal void AtualizaQuantidade(int quantidade)
        {
            Quantidade = quantidade;
         }
    }
```

De volta ao `ItemPedidoRepository`, teremos alterações feitas em memória, portanto precisamos descarregar essas alterações no banco dados. Para isso, utilizaremos o `contexto.SaveChanges()`.

Feito isso, adicionaremos breaking points na linha `if (itemPedidoDB != null)` e `itemPedidoDB.AtualizaQuantidade (itemPedido.Quantidade);`. Adicionaremos outro breaking point na linha `Quantidade = quantidade;` do método `AtualizaQuantidade()`.

```
public class ItemPedidoRepository: BaseRepository<ItemPedido>, IItemPedidoRepository
{
    public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
    {
    }

    public void UpdateQuantidade(ItemPedido itemPedido)
    {
        var itemPedidoDB =
        dbSet
            .Where(ip => ip.Id == itemPedido.Id)
            .SingleOrDefault();

        if (itemPedidoDB != null)
        {
            itemPedidoDB.AtualizaQuantidade (itemPedido.Quantidade);

            contexto.SaveChanges();
        }
```

Iremos executar a aplicação e, como de praxe, clicaremos sobre um item qualquer e depois sobre o botão "+" quando estivermos na página do carrinho. Quando essas ações forem finalizadas, seremos direcionados para a `PedidoController`. Veremos que para `itemPedido` teremos as seguintes informações:

```
Id : 12009
Pedido: null
PrecoUnitario : 0
Produto: null
Quantidade : 1
```

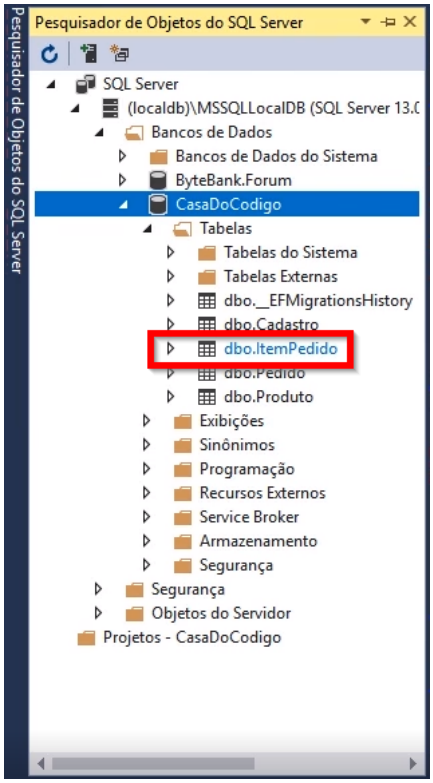
Pressionaremos o atalho "F5" para navegarmos até o próximo método. `ItemPedidoDB`, que foi carregado do banco de dados, terá as seguintes informações:

```
Id: 12009
Pedido: null
PrecoUnitario : 49,90
Produto: null
Quantidade : 1
```

Em seguida iremos até o método `AtualizaQuantidade()`, e veremos que a `quantidade` continua sendo `1`. Devemos inserir um valor que nos permita acompanhar se houve mudança no banco de dados, alteraremos o valor de `quantidade` para `117`.

```
internal void AtualizaQuantidade(int quantidade);
{
    Quantidade = quantidade;
}
```

Pressionaremos "F5" e iremos verificar se de fato houve mudança no banco de dados. Na área "Pesquisador de Objetos do SQL Server", clicaremos sobre "CasaDoCódigo > Tabelas > dbo.ItemPedido", clicaremos sobre o arquivo e selecionaremos a opção "Exibir Dados".




Como podemos ver por meio da tabela, os dados foram atualizados corretamente.

| Id | PedidoId | PrecoUnitario | ProdutoId | Quantidade |
|---|---|---|---|---|
| 1 | 1 | 49,90 | 54 | 1 |
| 2 | 1 | 49,90 | 3 | 1 |
| 3 | 1 | 49,90 | 65 | 1 |
| 4 | 1 | 49,90 | 37 | 1 |
| 5 | 2 | 49,90 | 2 | 1 |
| 6 | 2 | 49,90 | 1 | 1 |
| 7 | 2 | 49,90 | 3 | 1 |
| 8 | 3 | 49,90 | 3 | 1 |
| 9 | 4 | 49,90 | 1 | 1 |
| 1009 | 1004 | 49,90 | 2 | 1 |
| 2009 | 2004 | 49,90 | 2 | 1 |
| 3009 | 3004 | 49,90 | 2 | 1 |
| 4009 | 4004 | 49,90 | 2 | 1 |
| 5009 | 5004 | 49,90 | 1 | 1 |
| 6009 | 6004 | 49,90 | 14 | 1 |
| 7009 | 7004 | 49,90 | 2 | 1 |
| 8009 | 8004 | 49,90 | 1 | 1 |
| 9009 | 9004 | 49,90 | 2 | 1 |
| 10009 | 10004 | 49,90 | 1 | 1 |
| 11009 | 11004 | 49,90 | 2 | 1 |
| 12009 | 12004 | 49,90 | 2 | 177 |
| NULL | NULL | NULL | NULL | NULL |