Ctrl + ; = comenta TrustServerCertificate=True"

"Server=localhost,1433; Database=balta; User ID=sa; Password=1q2w3e4r@#$;” TrustServerCertificate=True";

Alt + Shit + F = Identação  
Alt + Seta pra cima = carrega linha  
Alt + seta pra baixo = carrega pra baixo  
Ctl +Sht + Enter = pula linha de cima  
Alt + SShit + seta pra baixo = duplicado linha  
Ctrl + D = Linha se repete  
Ctl + B = Barra do lado abre e fecha  
  
Pacotes  
dotnet add package Microsoft.EntityFrameWorkCore.SqlServer  
dotnet add package Microsoft.EntityFrameWorkCore

MODELS X ENTITY

As duas representam objetos que vao para o banco de dados, mas o Intity, possuem pontos a mais.

TABELA X COLUNA DA TABELA

Tela de celular com fundo preto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Data Contex é a representação do bando de dados em memória, possuem os DBset e refletem as tabelas.  
  
Criado pasta e arquivo DataContex.CS

Mapeamento das Categorias D = Para

Seu local

1º criado categoria em model  
Texto

Descrição gerada automaticamente

2º criado pasta Data – data contex representa o bando de memoria e Categories representa tabela de categorias  
Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

3º Em programa já é possível instanciar a categoria e fazer o CRUDE  
Texto

Descrição gerada automaticamente

* **Configurando o DataContext**

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

* **Create**
* CRUDE

Texto

Descrição gerada automaticamente

ToList

Para Ler qualquer informação do banco chama o var tags = contex.Tags; ( ainda não esta executando a Query no banco)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

var tags = context.Tags; -- aqui não executou no banco (aqui é apenas referencia)

foreach (var tag in tags) -- executou a query aqui no banco

{

Console.WriteLine(tag.Name);

}

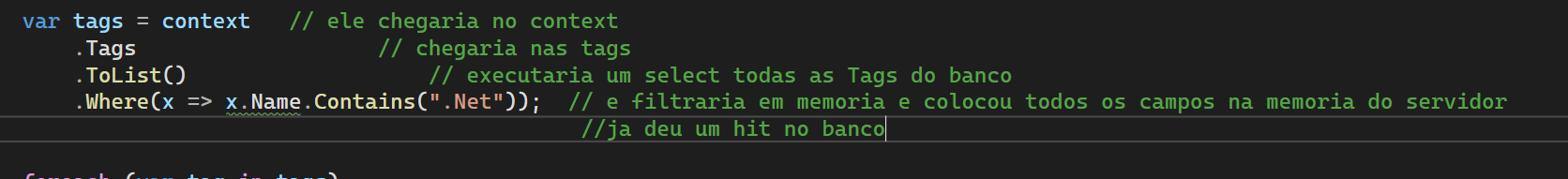
Nomalmente chamamos o .ToList(); ( forma a tags a serem executas, forca a query)  
  
Porem a ordem é importante.

Em filtros dinamentros por exemplo, filtro por nome.

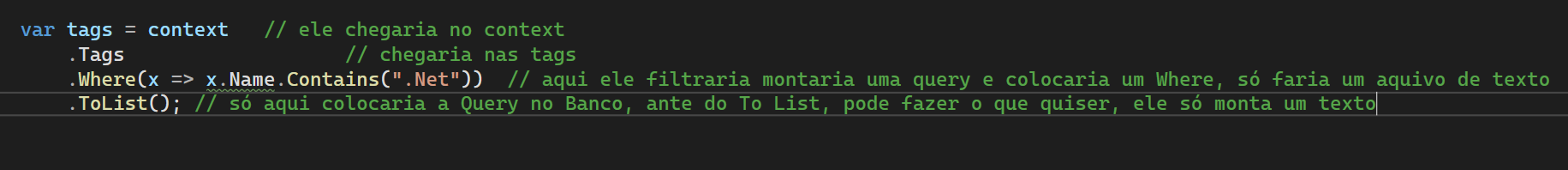
Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente

Aqui é um filtro que diz, me trás todas as tags que contem o nome .Net



Correto To List SEMPRE NO FINAL DA QUERY



AsNoTracking

Lendo um item só

var tag = context.Tags.FirstOrDefault(x => x.Id == 1021); criar ou excluir nao usar o new

Metadados são informações adicionais ao TAG, se não for remover ou Atualizar informações, informamos para na Query não trazer metadados, ao o EntityFramework ganha performance.  
Importante saber onde colocar .ToList .OrDefault e importante usar o AsNoTracking ele não vai traquear a Query

Para ler com foreach etc. sempre ler com asnotracking

Texto

Descrição gerada automaticamente

First, Single

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Single

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Mapeamento

Fluent Maps, separa o que é Cor, compartilhado entre as demais aplicações do Entity Framework das demais aplicações, quando existir aplicações para múltiplos dispositivos, vai existir a necessidade do fluente map

Texto

Descrição gerada automaticamente

Criando tabela

Texto

Descrição gerada automaticamente

Criando Indices

Texto

Descrição gerada automaticamente

Iniciando o mapeamento do user – sendo que algum deles tem relacionamento de N: p N:

Post possui N Tags,   
1 Tag pode estar em N Posts

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

No banco de dados terá uma tabela chamada PostTag, aqui não precisamos criar uma tabela associativa

Tela de computador com fundo preto

Descrição gerada automaticamente

Tela de computador com fundo preto

Descrição gerada automaticamente

Mapeamento - Relacionamento um para muitos

// Relacionamentos - Post e Autor - Post tem 1 Autor e Post tem 1 categoria - UM PARA MUITOS

1 Post tem 1 Categoria  
1 Categoria por conter muitos Posts  
1 Post tem 1 Autor  
1 Autor pode ter muitos Posts  
  
Em categoria, terá uma lista de Posts  
  
Em autor (User) tem uma lista de Post

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Mapeamento - Relacionamento muitos para muitos

Texto

Descrição gerada automaticamente

Tela de computador com fundo preto

Descrição gerada automaticamente

utilizando os mapeamentos criados

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Tela de computador

Descrição gerada automaticamente

Criando uma migração

dotnet ef database update \*\* atualizando banco

após atualizar fazer o

ef migrations

e depois o ef update e depois

dotnet run para iserir os dados

Texto

Descrição gerada automaticamente

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

dotnet ef migrations add NOMEDAATUALIZACAO - para add  
dotnet ef database update – para descer ao banco  
dotnet run – para inserir os dados se cadastrados

dotnet ef migrations remove - para remover a ultima atualização ( fazer isso ate chegar na versão desejável)  
dotnet ef database update – para descer ao banco

Gerando script do banco

Deve deletar a pasta Migrations no VS  
  
No banco dar um DROP TABLE \_\_EFMigrationsHistory -   
  
BD continua ok.  
  
Caso não tenha acesso ao Banco e não possa dar um donet ef database update, não tem acesso para create table, alter table, mas é preciso gerar um scrip pra alguém aplicar.  
  
Pode acessar via dotnet ef migrations script -o ./script.sql

Texto

Descrição gerada automaticamente  
  
Texto

Descrição gerada automaticamente  
  
Se colar esse script no BD ele criar um banco novo

[Performance e Outros](https://balta.io/player/assistir/dbb3fad9-8c65-4509-929f-b5fbb2622e8e/a0ca011a-3d01-42de-b06a-0292bb21929d)

AsNoTracking

Entity Framework ele é um OIM – OBJECT RELACIONAL MAP – OBJETO DE MAPEAMENTO RELACIONAL ele trata do CRUDE pra gente.  
  
é um recurso de performace, descasa as informações adicionacionais  
  
Se não for alterar, incluir, deletar, informações, não precisa usar o AsNoTracking  
Se for um Select simples

Async e Await

static async Task Main (string[] args)

{

using var context = new BlogDataContext();

var posts = context.Posts.FirstOrDefaultAsync(x => x.Id == 1);

var tags = context.Posts.FirstOrDefault (x => x.Id == 2);

Console.WriteLine("teste");

}

}

}

Async o código deixa de ser asincromo e vira Task tarefas asincromas.  
  
Texto

Descrição gerada automaticamente

IMPORTANTE

Texto

Descrição gerada automaticamente

using Azure;

using Blog.Data;

using Blog.Models;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System;

namespace Blog

{

class Program

{

static async Task Main(string[] args)

{ // 1º o cod. vai em contexto, vai buscar o contexto,

using var context = new BlogDataContext();

// 2º Ele vai buscar os Posts, nao vai esperar o Post terminar

var post = await context.Posts.ToListAsync();

var tags = await context.Users.ToListAsync(); // 3º e Ja vai Buscar os Tags

// para aguardar a execucao de tudo e dar o console.wriline por exemplo pode usar o Await

var posts = await GetPosts(context);

Console.WriteLine("teste");

// mesmo usando o await esperando para terminar e deixando assincromo, o sistema esta usando Treads diferentes, rodando

// informacoes paralelamente

}

// METODOS RETORNANDO VALORES ASYNC AWAIT ( no caso os posts dentro dele)

//1º criar metodo -

//2º async -

//3º Task -

//4º Tipo <List ou IEnume> <> -

//5º Classe retorno - <> dentro do tipo no caso list -

//6º Nome do Método

//7º receber o DB e variavel no caso context

//8º return

//9º await

//10º contex.

//11º.varial no pulural e primeira maiuscula.

//12.ToListAsync()

// vai retornar um IEnumerabele depois de Task

//13º charmar o metedoto para expor o resultado var posts = await GetPosts(context);

public static async Task<IEnumerable<Post>> GetPosts(BlogDataContext context)

{

return await context.Posts.ToListAsync();

}

}

}

Eager Loading VS Lazy Loading – Carregamento preguiçoso  
  
Texto

Descrição gerada automaticamente

Skip, Take e Paginação de dados

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Mapeando Queries Puras e Views

Cria uma classe em Models

Tela de computador com fundo preto

Descrição gerada automaticamente

Criar builder especial  
  
Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente  
  
Mapeando Viwers