ASP.NET Core MVC 2
- Entity Framework -

Curso de Ciência da Computação

- Programação II -

Prof. Mauricio Roberto Gonzatto


Entity Framework
Core

Entity Framework

- Ferramenta da Microsoft que permite ao desenvolvedor trabalhar com banco de dados relacional de maneira abstrata;
- Framework ORM (Object-Relational Mapping)
- Sua primeira versão saiu em 2008 no .NET Framework 3.5 SP1;

Entity Framework

- Se estamos utilizando uma base de dados relacional, haverá uma diferença entre os dados e as classes do domínio da aplicação.

- Usamos ORM para eliminar esta diferença e ignorar os problemas de persistência de dados, focando no código da aplicação e otimizando as questões de SQL – Structured Query Language.

Entity Framework

- Framework ORM – Object Relational Mapping, que possibilita aos desenvolvedores trabalhar nos objetos específicos do domínio da aplicação para acesso aos dados ao invés de trabalhar em consultas de banco de dados. Isto reduz a complexidade dos códigos na camada de acesso aos dados da aplicação.

ORM

- Abreviação para Object-Relational Mapping (mapeamento objeto relacional).

- Frameworks que possibilitam mapear as tabelas do banco de dados com classes de objeto.

- Se você possui uma tabela de Clientes no banco de dados, o ORM faz um mapeamento desta tabela com uma classe Cliente no seu projeto.

Conexão ao BD

- Existem basicamente 2 maneiras de conectar ao BD:
  - ADO.NET (Acess Data Object): Por meio das classes DbConnection, DbCommand, etc, podemos escrever manualmente os SQL;
  - ORM: Abstrai todo o processo do ADO.NET e cria os comandos SQL automaticamente;

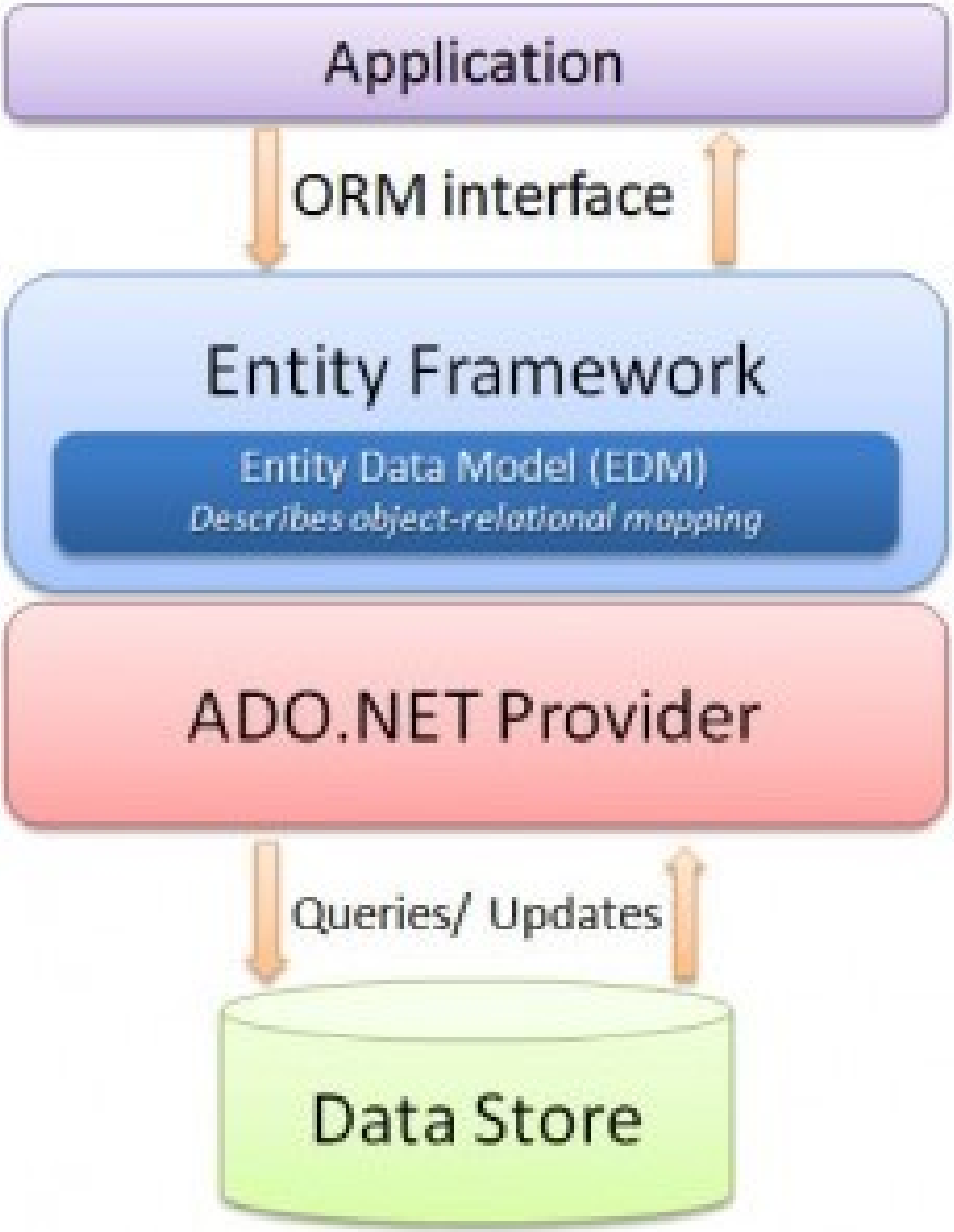
- Existem outros ORMs para .NET, como o Nhibernate que é uma versão derivada do Hibernate do Java. Mais informações: https://www.devmedia.com.br/analise-dos-melhores-orm-object-relational-mapping-para-plataforma-net/5548

Exemplo código ADO.NET

```
strConn = "string de conexão aqui";
SqlConnection objCon = new SqlConnection(strConn);
SqlCommand objComand = objCon.CreateCommand();
objComand.CommandText = "insert into Usuario
    (FirstName , LastName ) values (@FirstName ,
    @LastName )";
objComand.CommandType = CommandType.Text;
objComand.Parameters.Add("@FirstName  ",
    txtFirstName.Text );
objComand.Parameters.Add("@LastName  ",
    txtLastName.Text );
objComand.ExecuteNonQuery();
```

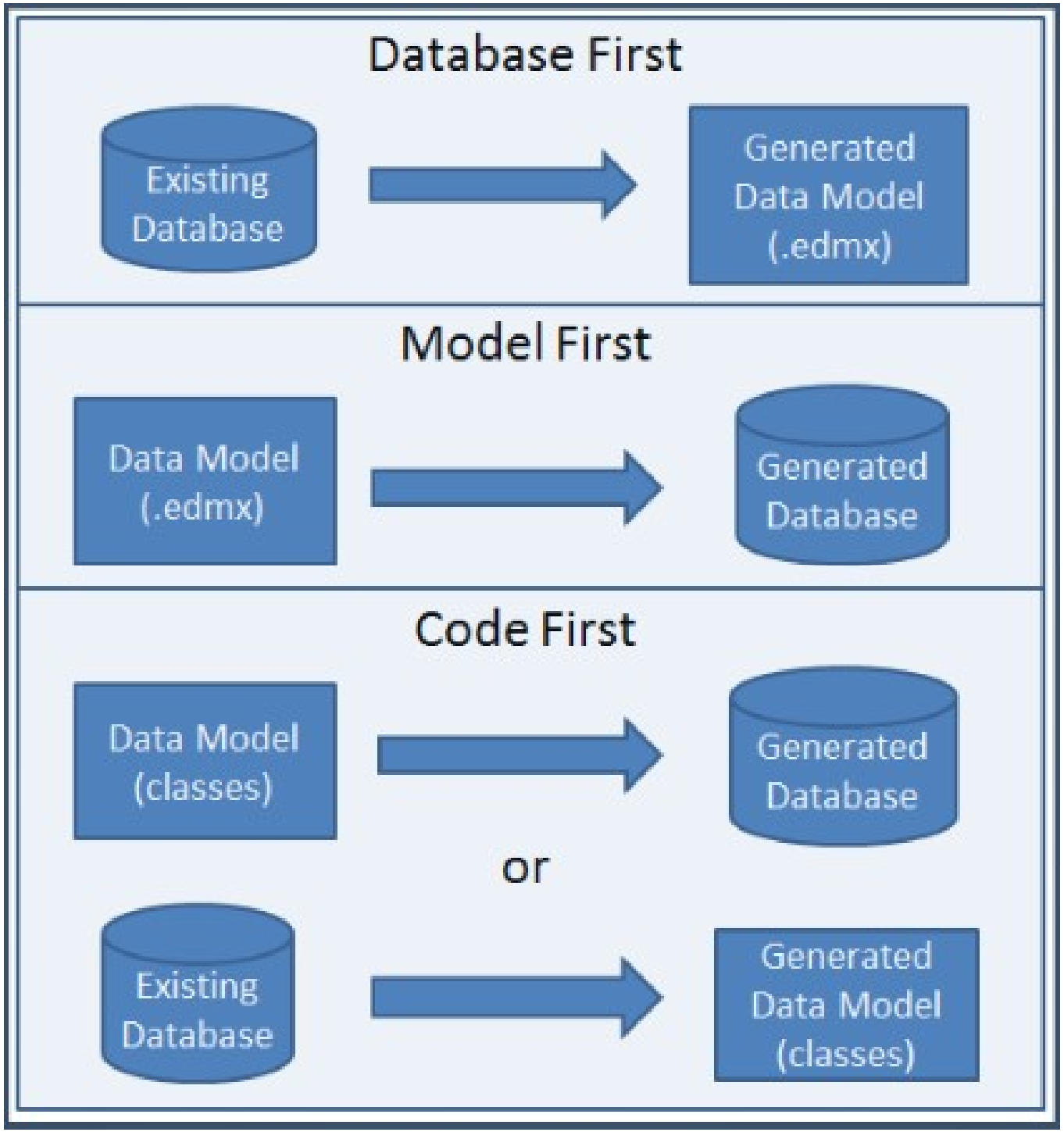
Exemplo código ORM

```
Usuario.AddNew();
Usuario.FirstName = this.txtFirstName.Text;
Usuario.LastName = this.txtLastName.Text;
Usuario.Save();
```


Application
ORM interface
Entity Framework
Entity Data Model (EDM)
Describes object-relational mapping
ADO.NET Provider
Queries/ Updates
Data Store

Como Funciona o EF?

- 03 formas de trabalhar com EF
  - **Database-First:** O banco de dados já existe, as classes são criadas baseadas nele;
  - **Model-First:** O banco de dados não existe, é criado um modelo conceitual e através dele é gerado um script para criação;
  - **Code-First:** Talvez a forma que melhor utiliza os recursos do EF. O Banco de dados ainda não existe, são criadas as classes simples, através delas toda a lógica da entidade é criada e com este modelo é gerado o BD.


Database First
Existing Database
Generated Data Model (.edmx)
Model First
Data Model (.edmx)
Generated Database
Code First
Data Model (classes)
Generated Database
or
Existing Database
Generated Data Model (classes)

Database First

- Em alguns cenários nos quais já existe um Banco de Dados podemos aplicar uma abordagem DataBase First.

- Podemos usar a ferramenta de **Scaffolding** e partindo do Banco de dados e suas tabelas, geramos o modelo de entidades de contexto.

Database First

- O comando usado para gerar o modelo de entidades a partir do banco de dados é :

dotnet ef dbcontext scaffold <string de conexão> Provider -o Models -f -c DemoDbContext

- Onde :
  - dotnet ef dbcontext - comando
  - <string de conexão> - a string de conexão do banco de dados usado
  - Provider - o provedor do banco de dados
  - -o Models - a pasta de sáida das classes geradas
  - -f - sobrescreve um código anteriormente gerado
  - -c DemoDbContext - o nome do DbContext usado na aplicação

Code-First

- Usado quando já possui uma aplicação ou código existente que modela uma estrutura de banco de dados;

- Código constituído por uma ou mais classes que definem o modelo;

Code First

```
public class Livro {
    // Define os campos usados no banco de dados
    public Int32 LivroID { get; set; }
    public String Nome { get; set; }
    public String Autor { get; set; }
    public String ISBN { get; set; }
    public Int32 Paginas { get; set; }
    public DateTime DataPublicacao { get; set; }
}
```

Code First

- O Fluxo de trabalho **Code First** foi criado pela Microsoft após os workflows **Model First** e
- para desenvolvedores que desejam escrever código, ao invés de trabalhar com um designer.

Code First

- Objetos do modelo são criados usando os objetos da CLR – Common Language Runtime com uma técnica de POCO (Plain Old C# Objects);

- Necessário a classe que serve de intermédio entre o modelo de classes e o DB. Ela é responsável por processar o mapeamento objeto-relacional e fazer com que os dados trafeguem para ambos os lados: domínio e banco;

Tipo de Banco de dados	Tipo de Projeto	Tipo de WorkFlow	Descrição
Banco de dados Novo	**Centrado no Designer**	**Model First**	Quando você precisa criar um banco de dados novo e que ver o projeto no formato gráfico. Neste fluxo você : • Cria o modelo usando o designer .EDMX • Informa ao Entity Framework para criar o banco de dados com base no seu projeto • O Entity Framework gera automaticamente as classes requeridas para interagir com o banco de dados
Banco de dados Existente	**Centrado no Designer**	**DataBase First**	Quando você já possui um banco de dados e deseja ver o projeto no formato gráfico. Neste fluxo você: • Informa ao Entity Framework para faze a engenharia reversa a partir do banco de dados existente para criar o modelo .EDMX • O Entity Framework gera automaticamente as classes requeridas para interagir com o banco de dados
Banco de dados Novo	**Centrado no código**	**Code First**	Quando você precisa criar um banco de dados novo e deseja ver o projeto no formato de código. Neste Fluxo você: • Define as classes que especificam os dados para cada tabela no banco de dados via código • Define o mapeamento usando classes • Opcionalmente define qualquer condição especial para criar o banco de dados e sua conexão • O Entity Framework gera o banco de dados em tempo de execução a partir das suas classes
Banco de dados Existente	**Centrado no código**	**Code First**	Quando o banco de dados já existe e você define código para mapear para os dados existentes. Neste Fluxo você: • Você realiza a engenharia reversa das classes que especificam os dados para cada tabela no banco de dados via código • Você faz a engenharia reversa do mapeamento criando classes que usam as definições das tabelas Nota: A Microsoft não fornece ferramentas para ajudar na engenharia reversa das bases de dados, para isso você precisa baixar o Entity Framework Power Tools.

```
public class BooksContext : DbContext
{
    public BooksContext()
    { }

    Public BooksContext
        (DbContextOptions<PersisteContext> opcoes)
    :base(opcoes) { }

    public DbSet<Livro> Livros { get; set; }

}
```

Criando um BD

```
private async Task CreateTheDatabaseAsync()
{
  using (var context = new BooksContext())
  {
    bool created = await context.Database.EnsureCreatedAsync();
    string creationInfo = created ? "created" : "exists";
    Console.WriteLine($"database {creationInfo}");
  }
}
```

Excluindo um BD

```
private async Task DeleteDatabaseAsync()
{
  Console.Write("Delete the database? ");
  string input = Console.ReadLine();
  if (input.ToLower() == "y")
  {
    using (var context = new BooksContext())
    {
      bool deleted = await context.Database.EnsureDeletedAsync();
      string deletionInfo = deleted ? "deleted" : "not deleted";
      Console.WriteLine($"database {deletionInfo}");
    }
  }
}
```

Gravando um registro

```
private async Task AddBookAsync(string title, string publisher)
{
  using (var context = new BooksContext())
  {
    var book = new Book
    {
      Title = title,
      Publisher = publisher
    };
    await context.Books.AddAsync(book);
    int records = await context.SaveChangesAsync();
    Console.WriteLine($"{records} record added");
  }
  Console.WriteLine();
}
```

Lendo dados do BD

```
private async Task ReadBooksAsync()
{
  using (var context = new BooksContext())
  {
    List<Book> books = await context.Books.ToListAsync();
    foreach (var b in books)
    {
      Console.WriteLine($"{b.Title} {b.Publisher}");
    }
  }
  Console.WriteLine();
}
```

Atualizando um registro

```
private async Task UpdateBookAsync()
{
  using (var context = new BooksContext())
  {
    int records = 0;
    Book book = await context.Books
      .Where(b => b.Title == "Professional C# 7")
      .FirstOrDefaultAsync();

    if (book != null)
    {
      book.Title = "Professional C# 7 and .NET Core 2.0";
      records = await context.SaveChangesAsync();
    }
    Console.WriteLine($"{records} record updated");
  }
  Console.WriteLine();
}
```

Excluindo registros

```
var books = context.Books;
context.Books.RemoveRange(books);
int records = await context.SaveChangesAsync();
Console.WriteLine($"{records} records deleted");
```

LINQ

- O Entity Framework Core usa LINQ (Consulta Integrada à Linguagem) para consultar dados do banco de dados. O LINQ permite que você use C# (ou a linguagem .NET de sua escolha) para escrever consultas fortemente tipadas com base em suas classes derivadas de contexto e entidade.

LINQ

- Uma representação da consulta LINQ é passada para o provedor de banco de dados, para ser convertida em linguagem de consulta específica do banco de dados (por exemplo, SQL para um banco de dados relacional).

- O Entity Framework Core pode acessar muitos bancos de dados diferentes por meio de bibliotecas de plug-in chamadas de provedores de banco de dados.

  - Lista de provedores disponíveis:
  - https://docs.microsoft.com/pt-br/ef/core/providers/index

Carregar todos os dados

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs.ToList();
}
```

Carregar uma única entidade

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);
}
```

Filtragem

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Where(b => b.Url.Contains("dotnet"))
        .ToList();
}
```

Dados Relacionados

- O Entity Framework Core permite que você use as propriedades de navegação em seu modelo para carregar as entidades relacionadas. Há três padrões de O/RM comuns usados para carregar os dados relacionados.
  - **Carregamento adiantado** significa que os dados relacionados são carregados do banco de dados como parte da consulta inicial.
  - **Carregamento explícito** significa que os dados relacionados são explicitamente carregados do banco de dados em um momento posterior.
  - **Carregamento lento** significa que os dados relacionados são carregados de modo transparente do banco de dados quando a propriedade de navegação é acessada.

Carregamento Adiantado

- Você pode usar o método Include para especificar os dados relacionados a serem incluídos nos resultados da consulta. No exemplo a seguir, os blogs que são retornados nos resultados terão suas propriedades Posts preenchidas com as postagens relacionadas.

Carregamento Adiantado

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .ToList();
}
```

Incluindo vários níveis

- Você pode fazer uma busca detalhada por meio de relações para incluir vários níveis de dados relacionados usando o método ThenInclude. O exemplo a seguir carrega todos os blogs, suas postagens relacionadas e o autor de cada postagem.

Incluindo vários níveis

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
        .ToList();
}
```

Dados relacionados

- Mais detalhes e exemplos de consultas e manipulação de dados relacionados direto na documentação do EF:

- https://docs.microsoft.com/pt-br/ef/core/querying/related-data

Migrations

- Um modelo de dados muda durante o desenvolvimento e fica fora de sincronia com o banco de dados. Você pode descartar o banco de dados e permitir que o EF crie um novo que corresponda ao modelo, mas esse procedimento resulta na perda de dados. O recurso de migrações no EF Core oferece uma maneira de atualizar de forma incremental o esquema de banco de dados para mantê-lo em sincronia com o modelo de dados do aplicativo, preservando os dados existentes no banco de dados.

Migrations

- As migrações incluem ferramentas de linha de comando e APIs que ajudam com as seguintes tarefas:

- **Criar uma migração.** Gerar um código que pode atualizar o banco de dados para sincronizá-lo com um conjunto de alterações do modelo.

- **Atualizar o banco de dados.** Aplicar migrações pendentes para atualizar o esquema de banco de dados.

- **Personalizar o código de migração.** Às vezes, o código gerado precisa ser modificado ou complementado.

Migrations

- **Remover uma migração.** Excluir o código gerado.
- **Reverter uma migração.** Desfazer as alterações do banco de dados.
- **Gerar scripts SQL**. Talvez seja necessário um script para atualizar um banco de dados de produção ou para solucionar problemas de código de migração.
- **Aplicar migrações em tempo de execução.** Quando as atualizações de tempo de design e a execução de scripts não forem as melhores opções, chame o método Migrate().

Criar Migração

- dotnet ef migrations add InitialCreate

- Três arquivos são adicionados ao seu projeto no diretório **Migrações**:
  - **00000000000000_InitialCreate.CS**– o arquivo principal de migrações. Contém as operações necessárias para aplicar a migração (em Up()) e revertê-la (em Down()).
  - **00000000000000_InitialCreate.Designer.CS**– o arquivo de metadados de migrações. Contém informações usadas pelo EF.
  - **MyContextModelSnapshot.cs**– um instantâneo do seu modelo atual. Usado para determinar o que mudou ao adicionar a próxima migração.

- O carimbo de data/hora no nome de arquivo ajuda a mantê-lo organizado por ordem cronológica para que você possa ver o andamento das alterações.

Remover Migração

- Às vezes, você adiciona uma migração e percebe que precisa fazer alterações adicionais ao modelo do EF Core antes de aplicá-lo. Para remover a última migração, use este comando. Após remover a migração, você poderá fazer as alterações adicionais ao modelo e adicioná-la novamente.

- dotnet ef migrations remove

Reverter Migração

- Se você já tiver aplicado uma migração (ou várias migrações) no banco de dados, mas precisar revertê-la, poderá usar o mesmo comando usado para aplicar as migrações, mas especificando o nome da migração para a qual você deseja reverter.

- dotnet ef database update LastGoodMigration

Atualizar BD

- dotnet ef database update

- Mais informações sobre migrações podem ser encontradas na documentação do EF:
- https://docs.microsoft.com/pt-br/ef/core/managing-schemas/migrations/