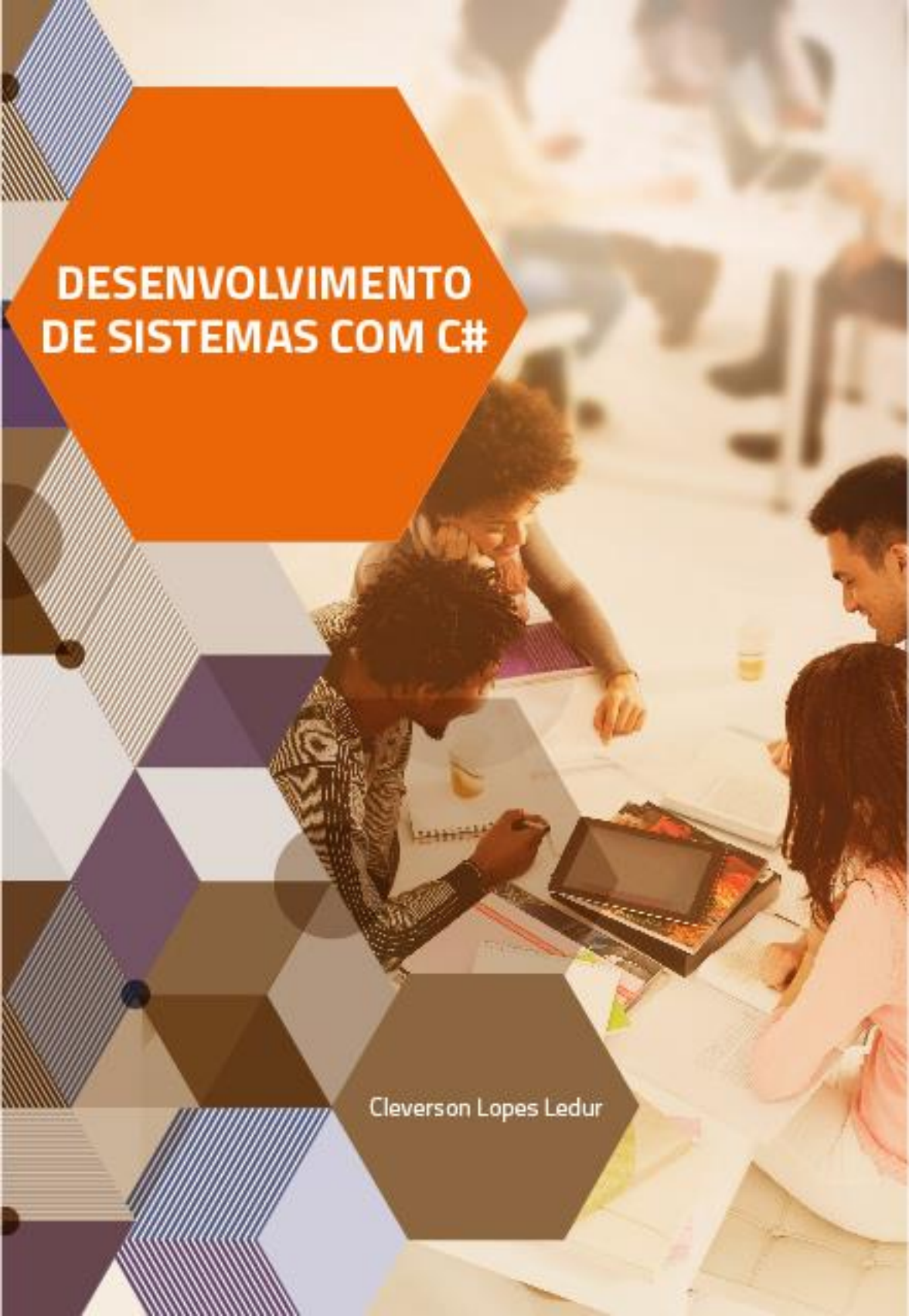
DESENVOLVIMENTO DE SISTEMAS COM C#



Cleverson Lopes Ledur

Revisão técnica:

Jeferson Faleiro Leon
Desenvolvimento de Sistemas
Especialista Formação Pedagógica de Professores
Professor do curso Técnico em informática

L475d Ledur, Cleverson Lopes.
Desenvolvimento de sistemas com #C [recurso eletrônico] / Cleverson Lopes Ledur; [revisão técnica: Jeferson Faleiro Leon]. – Porto Alegre : SAGAH, 2018.

ISBN 978-85-9502-314-7

1. Ciência da computação. 2. Linguagens de programação de computador. I. Título.

CDU 004.43

Catalogação na publicação: Karin Lorien Menoncin CRB-10/2147

Utilizar o Entity Framework para persistência em banco de dados – II

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Explicar o conceito de migração no Entity Framework.
- Habilitar e adicionar migrações.
- Atualizar o banco de dados.

Introdução

Quando você desenvolve um novo sistema, seu modelo de dados muda com frequência. E, cada vez que o modelo se altera, ele fica fora de sincronia com o banco de dados. Se você já tiver configurado o Entity Framework em um projeto, quando você alterar os modelos ele se modificará automaticamente e recriará o banco de dados. Dessa forma, você poderá, em alguns casos, perder informações ou perceber um comportamento não esperado na estrutura do banco de dados. Para tornar essas operações mais controladas, foram criadas as migrações. Com elas, você pode trabalhar as alterações dos modelos e a sincronização do banco de dados como se fossem alterações de código, por meio de um sistema baseado em versionamento de código.

Neste texto, você vai entender qual é o conceito de migração no Entity Framework. Além disso, vai ver como habilitar e adicionar migrações e também como atualizar o banco de dados de uma aplicação.

Migrações

Quando você desenvolve um novo sistema, seu modelo de dados muda com frequência, não é? E toda vez que o modelo se altera, ele fica fora de sincronia

com o banco de dados. Se você já tiver configurado o Entity Framework (EF) em um projeto, quando você alterar os modelos ele se modificará automaticamente e recriará o banco de dados. Quando você adiciona, remove ou altera classes de entidade, bem como quando modifica sua classe `DbContext`, na próxima vez que executa a aplicação, o sistema exclui automaticamente o banco de dados existente, cria novas tabelas ou se altera de forma a combinar com o modelo e os atributos existentes (MICROSOFT, 2017).

Esse método de manter o banco de dados em sincronia com o modelo de dados funciona bem até que você implante o aplicativo na produção. Quando o aplicativo está sendo executado em produção (pelos usuários), ele geralmente armazena os dados que deseja manter. E você não quer perder tudo cada vez que faz uma alteração, como ao adicionar uma nova coluna, não é? O recurso de migrações resolve esse problema ativando o código de migração para atualizar o esquema do banco de dados. Assim, a atualização não é realizada automaticamente sem o seu consentimento (MICROSOFT, 2017).

Se você já utilizou um sistema de controle de versão como o SVN e o GIT, terá bastante facilidade no entendimento desses conceitos, já que eles utilizam os mesmos princípios. Basicamente, o EF detecta as mudanças no código dos modelos e as armazena localmente. O desenvolvedor, então, utiliza comandos que são passados pelo console para aplicar as alterações. Assim, um arquivo é gerado com essas alterações (mas ainda não é executado). Esse arquivo fica armazenado em um diretório do projeto com a extensão `.migration`.

Outro comando permitirá que esse arquivo seja executado e então o banco será alterado.

De forma geral, dentro do escopo de migrações, para todas as alterações que você fizer nas classes, executará basicamente dois comandos:

- `Add-Migrations`
- `Update-DataBase`

A seguir, você pode ver a estrutura de cada um desses comandos (ENTITY FRAMEWORK TUTORIAL, c2016).

- Add-Migrations "`nome _ migrations`" – cria uma alteração no banco de dados. O *`nome _ migrations`* é o nome que você irá dar para a atualização;
- `Update-DataBase` – aplica as alterações ao banco de dados;
- `Update-DataBase - script` – gera um script com os comandos SQL para serem executados no banco de dados.

Utilizando as migrações

Você deve utilizar as migrações como forma de obter mais controle sobre as alterações realizadas pelo Entity Framework. Nesta seção, você vai ver como utilizar as migrações em um projeto. Para utilizar as migrações na prática, você precisa primeiro criar um novo projeto do tipo **Console**. Coloque o nome que você preferir no projeto. No exemplo da Figura 1, foi utilizado o nome **ExemploMigrations**.

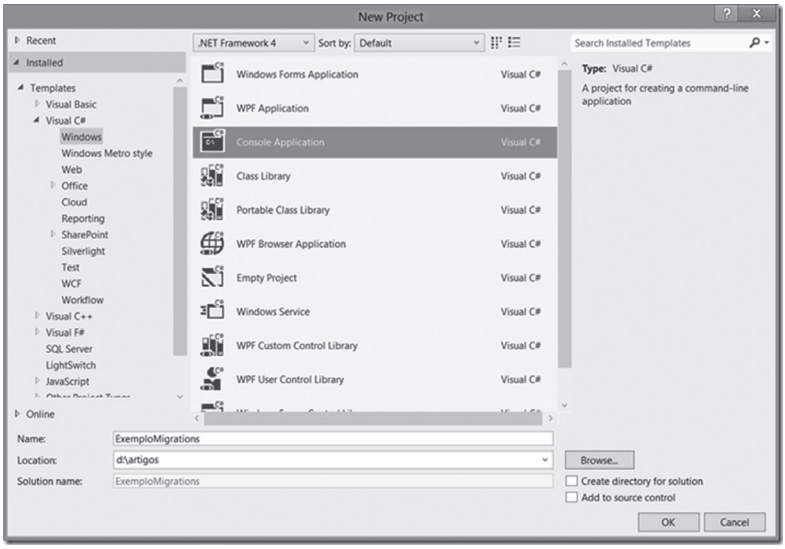


Figura 1. Crie um novo projeto e dê um nome a ele.

Inicialmente, você deve instalar o Entity Framework. Você pode fazer isso usando a interface gráfica, ou também pode realizar a instalação diretamente pela linha de comando. Para isso, abra **Tools > Library Package Manager > Packager Manager Console** e digite *Install-Package EntityFramework* (Figura 2).

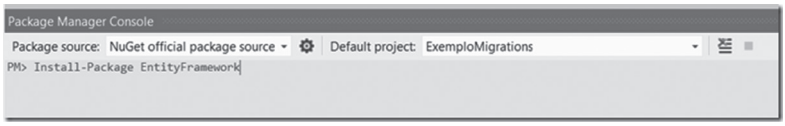


Figura 2. Na janela do **Packager Manager Console**, digite **Install-Package EntityFramework**.

Agora, comece criando a classe chamada `Contexto` e estenda a classe `DbContext`. Nela, você deve informar ao Entity Framework que deseja ter uma tabela no banco de dados para mapear a sua classe `Cliente`. Veja o exemplo a seguir.

```
public class Contexto : DbContext
    {
        public DbSet<Cliente> Cliente { get; set; }
    }
```

Agora, você precisa criar também a classe `Cliente`. Adicione dois atributos, `ID` e `Nome`. Também coloque os seus getters e setters. Observe o próximo exemplo.

```
public class Cliente
    {
        public int ID { get; set; }
        public string Nome { get; set; }
    }
```

Adicione também um arquivo **app.config** para identificar seu servidor SQL e o nome do banco de dados, como mostra o próximo exemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="Contexto" providerName="System.Data.SqlClient"
connectionString="data source=(local); initial catalog=ExemploMigrations;
integrated security=true;"/>
  </connectionStrings>
</configuration>
```

Agora que já tem o seu banco de dados configurado no arquivo **app.config**, você pode realmente começar a utilizar as migrações. Inicialmente, deve iniciar (habilitar) as migrações. Para isso, digite no console o seguinte comando:

PM> Enable-Migrations

Esse comando habilita as migrações no projeto. Agora, você pode adicionar uma das migrações. Digite o comando mostrado a seguir para criar a primeira migração.

```
PM> Add-Migration Inicial
```

Com isso, um novo diretório foi criado no projeto chamado *Migrations*. Nesse diretório, você tem o arquivo mostrado no próximo exemplo.

```
public partial class Inicial : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "Clientes",
                c => new
                    {
                        ID = c.Int(nullable: false, identity: true),
                        Nome = c.String(),
                    })
                .PrimaryKey(t => t.ID);

        }

        public override void Down()
        {
            DropTable("Clientes");
        }
    }
```

Repare que no arquivo mostrado você tem um `CreateTable`. Nele, você está inserindo a tabela *Clientes* no banco de dados e passando os atributos `ID` e `Nome` que serão criados. Você também está informando que a `PrimaryKey` será o seu ID.

Essa é uma classe `Migration`. Ela irá alterar o seu banco de dados. Até o momento, você ainda não fez nenhuma alteração. Logo, se você rodar a aplicação, é possível que um erro seja emitido. Para que as alterações da classe `Migration` sejam realizadas, você tem de rodar o comando:

```
PM> Update-DataBase
```

Atualizações no banco de dados

Cada vez que você atualizar a sua classe modelo, precisará rodar os comandos de migração do banco de dados para que as alterações se efetivem na estrutura

das tabelas. Por exemplo, se um novo atributo for adicionado em uma classe que está dentro do contexto do Entity Framework, você deve rodar os comandos de migração para garantir que a aplicação seja executada com os modelos e as tabelas mapeadas igualitariamente.

Outro caso em que você também deve executar os comandos de migração é na criação de novos modelos. Os comandos de migração devem ser executados para garantir que as novas tabelas sejam criadas.

A cada alteração, você deve executar no console os comandos mostrados a seguir.

```
PM> Add-Migration "NOME DA MIGRAÇÃO"
PM> Update-DataBase
```

Assim, você cria uma migração passando o nome da migração com o primeiro comando. Ela ficará armazenada no diretório de migrações, mas não irá alterar nada no seu banco de dados. Após, o Update-Database irá executar as migrações que ainda não foram aplicadas. Caso já existam migrações que foram executadas, elas não serão executadas novamente.

Saiba mais

Em alguns casos, você também pode alterar migrações. Um uso muito comum das alterações das migrações é quando se deseja criar uma nova tabela e já inserir alguns dados iniciais para colocar informações no sistema antes mesmo de ele ser utilizado. Isso ocorre, por exemplo, se o sistema utiliza uma tabela para armazenar os menus e outra onde ficam as permissões de tipos de usuários. Como essas informações mudam pouco, elas já poderiam ser inseridas por meio de uma *migration*.

Referências

ENTITY FRAMEWORK TUTORIAL. *Code-based migration*. [S.l.], c2016. Disponível em: <http://www.entityframeworktutorial.net/code-first/code-based-migration-in-code-first.aspx>. Acesso em: 22 nov. 2017.

MICROSOFT. *Entity framework code first migrations*. Redmond, 2017. Disponível em: <https://msdn.microsoft.com/en-us/library/jj591621(v=vs.113).aspx>. Acesso em: 27 nov. 2017.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo: