



K9

TREINAMENTOS

Desenvolvimento Web com ASP.NET MVC 4

Desenvolvimento Web com ASP.NET MVC 4

20 de fevereiro de 2013

Sumário	i
Sobre a K19	1
Seguro Treinamento	2
Termo de Uso	3
Cursos	4
1 Banco de Dados	1
1.1 Sistemas Gerenciadores de Banco de Dados	1
1.2 SQL Server	2
1.3 Bases de Dados (<i>Databases</i>)	2
1.4 Tabelas	3
1.5 CRUD	6
1.6 Chaves Primária e Estrangeira	9
1.7 Exercícios de Fixação	10
2 ADO.NET	23
2.1 Driver	24
2.2 ODBC	25
2.3 ODBC Manager	25
2.4 Criando uma conexão	26
2.5 Inserindo registros	26
2.6 Exercícios de Fixação	27
2.7 Exercícios Complementares	28
2.8 SQL Injection	28
2.9 Exercícios de Fixação	29
2.10 Exercícios Complementares	30
2.11 Listando registros	30
2.12 Exercícios de Fixação	31
2.13 Exercícios Complementares	32
2.14 Connection Factory	32

2.15 Exercícios de Fixação	33
2.16 Exercícios Complementares	35
2.17 Desafios	35
3 Entity Framework	37
3.1 Múltiplas sintaxes da linguagem SQL	37
3.2 Orientação a Objetos VS Modelo Relacional	37
3.3 Ferramentas ORM	38
3.4 Instalando o Entity Framework	39
3.5 Configuração	40
3.6 Convenções de Mapeamento	41
3.7 Exercícios de Fixação	47
3.8 Data Annotations	50
3.9 Exercícios de Fixação	55
3.10 Gerenciamento de entidades	55
3.11 Exercícios de Fixação	57
3.12 Repositórios	59
4 Visão Geral do ASP.NET MVC	61
4.1 Necessidades de uma aplicação web	61
4.2 ASP.NET MVC	62
4.3 MVC e Front Controller	62
4.4 Visual Studio	63
4.5 Exemplo de uma Aplicação Web	64
4.6 Exercícios de Fixação	65
4.7 Integração com Entity Framework	70
4.8 Scaffold	70
4.9 Exercícios de Fixação	71
5 Camada de Apresentação	75
5.1 Razor e ASPX	75
5.2 Exercícios de Fixação	77
5.3 Exercícios Complementares	78
5.4 ViewBag e Strogly Typed Views	79
5.5 Exercícios de Fixação	81
5.6 HTML Helpers	84
5.7 Exercícios de Fixação	93
5.8 Layouts	97
5.9 Exercícios de Fixação	100
5.10 Partial views	106
5.11 Exercícios de Fixação	108
6 Camada de Controle	111
6.1 Actions	111
6.2 ActionResult	112
6.3 Parâmetros	113
6.4 Exercícios de Fixação	114
6.5 TempData	117
6.6 Exercícios de Fixação	118
6.7 Rotas	119

6.8 Exercícios de Fixação	121
7 Validação	125
7.1 Controller	125
7.2 View	126
7.3 Exercícios de Fixação	127
7.4 Anotações	130
7.5 Validação no lado do Cliente	131
7.6 Exercícios de Fixação	132
8 Sessão	135
8.1 Identificando os navegadores	135
8.2 Sessões no ASP.NET MVC	135
8.3 Session Mode	136
8.4 Exercícios de Fixação	137
9 Autenticação	143
9.1 Filtro de Autenticação	143
9.2 Exercícios de Fixação	145
10 Tratamento de Erros	151
10.1 Try-Catch	152
10.2 Custom Errors	153
10.3 Http Errors	153
10.4 Exercícios de Fixação	154
A ASP.NET Web API	159
A.1 REST	159
A.2 Resources	160
A.3 URIs	160
A.4 Operações	160
A.5 Media Type	162
A.6 Exercícios de Fixação	162
B Migrations	167
B.1 Passo a Passo	167
B.2 Exercícios de Fixação	174
C Projeto	183
C.1 Modelo	183
C.2 Exercícios de Fixação	183
C.3 Persistência - Mapeamento	184
C.4 Exercícios de Fixação	184
C.5 Persistência - Configuração	185
C.6 Exercícios de Fixação	185
C.7 Persistência - Repositórios	185
C.8 Exercícios de Fixação	185
C.9 Exercícios de Fixação	187
C.10 Apresentação - Template	188
C.11 Exercícios de Fixação	188
C.12 Cadastrando e Listando Seleções	191

C.13 Exercícios de Fixação	191
C.14 Removendo Seleções	194
C.15 Exercícios de Fixação	194
C.16 Cadastrando, Listando e Removendo Jogadores	196
C.17 Exercícios de Fixação	196
C.18 Removendo Jogadores	200
C.19 Exercícios de Fixação	200
C.20 Membership e Autorização	202
C.21 Exercícios de Fixação	202
C.22 Exercícios de Fixação	210
C.23 Exercícios de Fixação	211
C.24 Controle de Erro	212
C.25 Exercícios de Fixação	212
C.26 Enviando email	213
C.27 Exercícios de Fixação	213
D Respostas	217



Sobre a K19

A K19 é uma empresa especializada na capacitação de desenvolvedores de software. Sua equipe é composta por profissionais formados em Ciência da Computação pela Universidade de São Paulo (USP) e que possuem vasta experiência em treinamento de profissionais para área de TI.

O principal objetivo da K19 é oferecer treinamentos de máxima qualidade e relacionados às principais tecnologias utilizadas pelas empresas. Através desses treinamentos, seus alunos se tornam capacitados para atuar no mercado de trabalho.

Visando a máxima qualidade, a K19 mantém as suas apostilas em constante renovação e melhoria, oferece instalações físicas apropriadas para o ensino e seus instrutores estão sempre atualizados didática e tecnicamente.



Seguro Treinamento

Na K19 o aluno faz o curso quantas vezes quiser!

Comprometida com o aprendizado e com a satisfação dos seus alunos, a K19 é a única que possui o Seguro Treinamento. Ao contratar um curso, o aluno poderá refazê-lo quantas vezes desejar mediante a disponibilidade de vagas e pagamento da franquia do Seguro Treinamento.

As vagas não preenchidas até um dia antes do início de uma turma da K19 serão destinadas ao alunos que desejam utilizar o Seguro Treinamento. O valor da franquia para utilizar o Seguro Treinamento é 10% do valor total do curso.



Termo de Uso

Termo de Uso

Todo o conteúdo desta apostila é propriedade da K19 Treinamentos. A apostila pode ser utilizada livremente para estudo pessoal . Além disso, este material didático pode ser utilizado como material de apoio em cursos de ensino superior desde que a instituição correspondente seja reconhecida pelo MEC (Ministério da Educação) e que a K19 seja citada explicitamente como proprietária do material.

É proibida qualquer utilização desse material que não se enquadre nas condições acima sem o prévio consentimento formal, por escrito, da K19 Treinamentos. O uso indevido está sujeito às medidas legais cabíveis.



Conheça os nossos cursos

-  K01 - Lógica de Programação
-  K02 - Desenvolvimento Web com HTML, CSS e JavaScript
-  K03 - SQL e Modelo Relacional
-  K11 - Orientação a Objetos em Java
-  K12 - Desenvolvimento Web com JSF2 e JPA2
-  K21 - Persistência com JPA2 e Hibernate
-  K22 - Desenvolvimento Web Avançado com JFS2, EJB3.1 e CDI
-  K23 - Integração de Sistemas com Webservices, JMS e EJB
-  K41 - Desenvolvimento Mobile com Android
-  K51 - Design Patterns em Java
-  K52 - Desenvolvimento Web com Struts
-  K31 - C# e Orientação a Objetos
-  K32 - Desenvolvimento Web com ASP.NET MVC

www.k19.com.br/cursos

BANCO DE DADOS

Em geral, as aplicações necessitam armazenar dados de forma persistente para consultá-los posteriormente. Por exemplo, a aplicação de uma livraria precisa armazenar os dados dos livros e dos autores de forma persistente.

Suponha que esses dados sejam armazenados em arquivos do sistema operacional. Vários fatores importantes nos levam a descartar tal opção. A seguir, apresentamos as principais dificuldades a serem consideradas na persistência de dados.

Segurança: O acesso às informações potencialmente confidenciais deve ser controlado de forma que apenas usuários e sistemas autorizados possam manipulá-las.

Integridade: Restrições relacionadas aos dados armazenados devem ser respeitadas para que as informações estejam sempre consistentes.

Consulta: O tempo gasto para realizar as consultas aos dados armazenados deve ser o menor possível.

Concorrência: Em geral, diversos sistemas e usuários acessarão concorrentemente as informações armazenadas. Apesar disso, a integridade dos dados deve ser preservada.

Considerando todos esses aspectos, concluímos que um sistema complexo seria necessário para persistir as informações de uma aplicação de maneira adequada. Felizmente, tal tipo de sistema já existe e é conhecido como **Sistema Gerenciador de Banco de Dados** (SGBD).

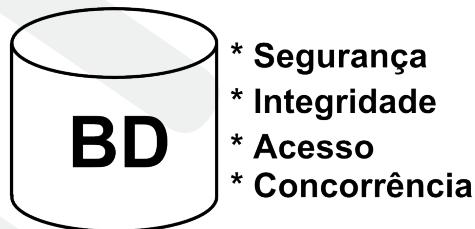


Figura 1.1: Sistema Gerenciador de Banco de Dados

Sistemas Gerenciadores de Banco de Dados

No mercado, há diversas opções de sistemas gerenciadores de banco de dados. Os mais populares são:

- Oracle Database
- SQL Server

- MySQL Server
- PostgreSQL

SQL Server

Neste treinamento, utilizaremos o SQL Server Express, que é mantido pela Microsoft. O SQL Server Express pode ser obtido a partir do site:

<http://www.microsoft.com/express/Database/>.

Microsoft SQL Server Management Studio Express

Para interagir com o SQL Server Express, utilizaremos um cliente com interface gráfica chamado de Microsoft SQL Server Management Studio Express.

Bases de Dados (*Databases*)

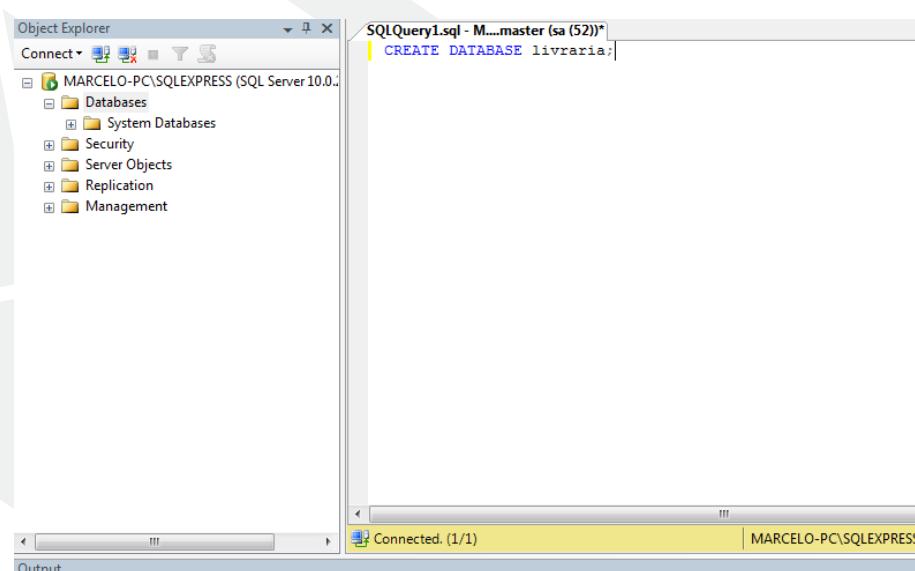
Um sistema gerenciador de banco de dados é capaz de gerenciar informações de diversos sistemas ao mesmo tempo. Por exemplo, as informações dos clientes de um banco, além dos produtos de uma loja virtual ou dos livros de uma livraria.

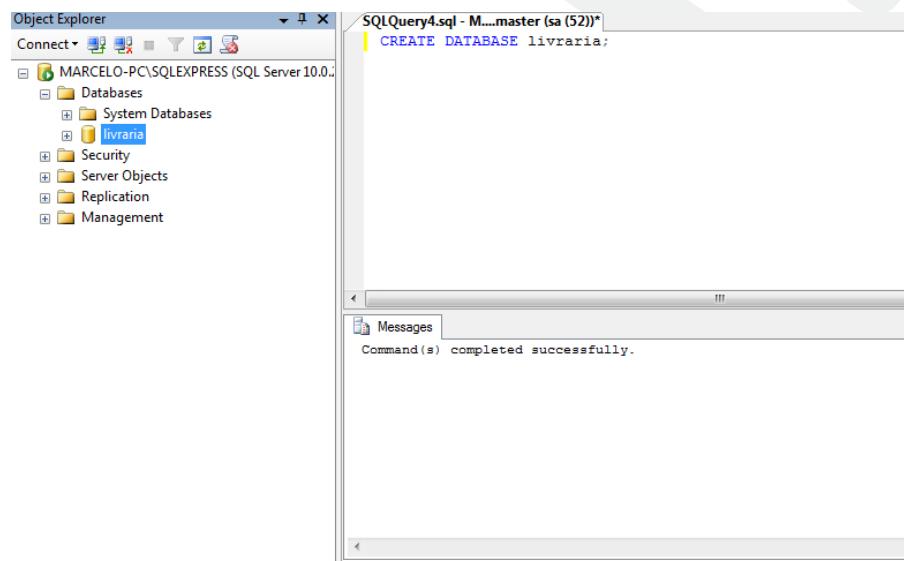
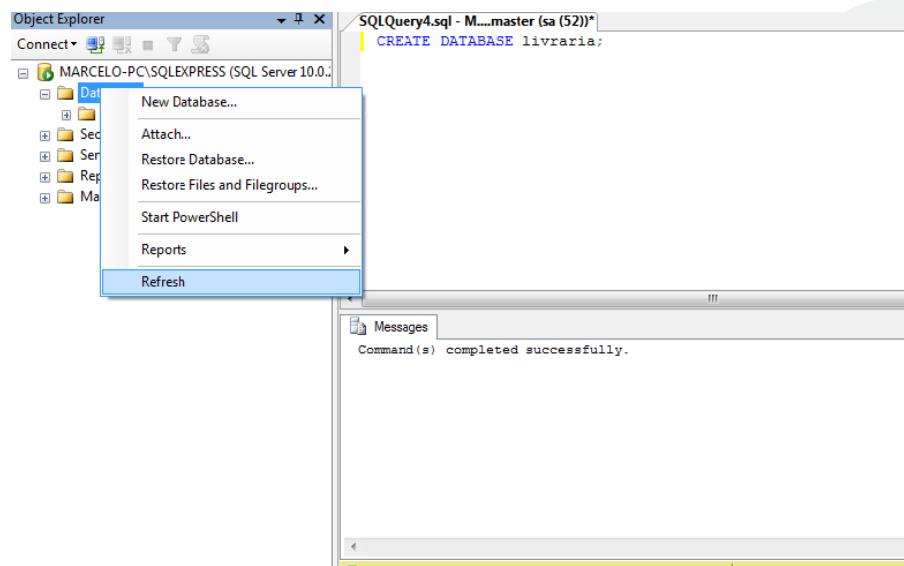
Suponha que os dados fossem mantidos sem nenhuma separação lógica. Implementar regras de segurança específicas seria extremamente complexo. Tais regras criam restrições quanto ao conteúdo que pode ser acessado por cada usuário. Por exemplo, determinado usuário poderia ter permissão de acesso aos dados dos clientes do banco, mas não às informações dos produtos da loja virtual, ou dos livros da livraria.

Para obter uma organização melhor, os dados são armazenados separadamente em um SGDB. Daí surge o conceito de **base de dados** (database). Uma base de dados é um agrupamento lógico das informações de um determinado domínio.

Criando uma base de dados no SQL Server Express

Para criar uma base de dados no SQL Server Express, utilizamos o comando **CREATE DATABASE**.





Repare que além da base de dados **livraria** há outras bases. Essas bases foram criadas automaticamente pelo próprio SQL Server Express para teste ou para guardar algumas configurações.

Quando uma base de dados não é mais necessária, ela pode ser removida através do comando **DROP DATABASE**.

Tabelas

Um servidor de banco de dados é dividido em bases de dados com o intuito de separar as informações de domínios diferentes. Nessa mesma linha de raciocínio, podemos dividir os dados de uma base a fim de agrupá-los segundo as suas correlações. Essa separação é feita através de **tabelas**. Por exemplo, no sistema de um banco, é interessante separar o saldo e o limite de uma conta, do nome e CPF de um cliente. Então, poderíamos criar uma tabela para os dados relacionados às contas e outra para os dados relacionados aos clientes.

Uma tabela é formada por **registros** (linhas) e os registros são formados por **campos** (colunas).

Cliente		
nome	idade	cpf
José	27	31875638735
Maria	32	30045667856

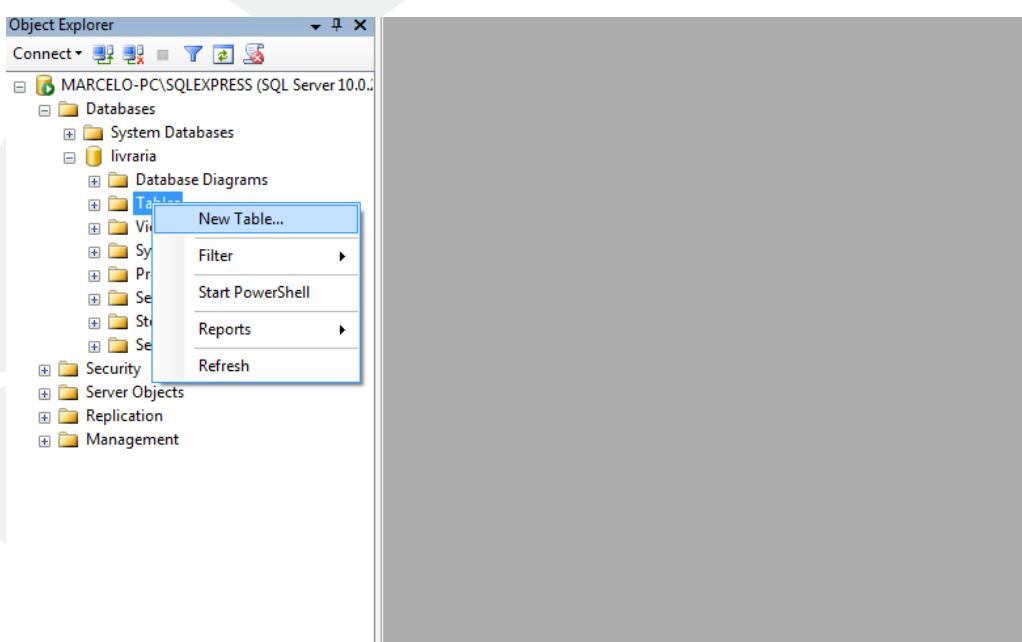
Conta		
numero	saldo	limite
1	1000	500
2	2000	700

Tabela 1.1: Tabelas para armazenar os dados relacionados aos clientes e às contas

Por exemplo, considere uma tabela para armazenar as informações dos clientes de um banco. Cada registro dessa tabela armazena em seus campos os dados de um determinado cliente.

Criando tabelas no SQL Server Express

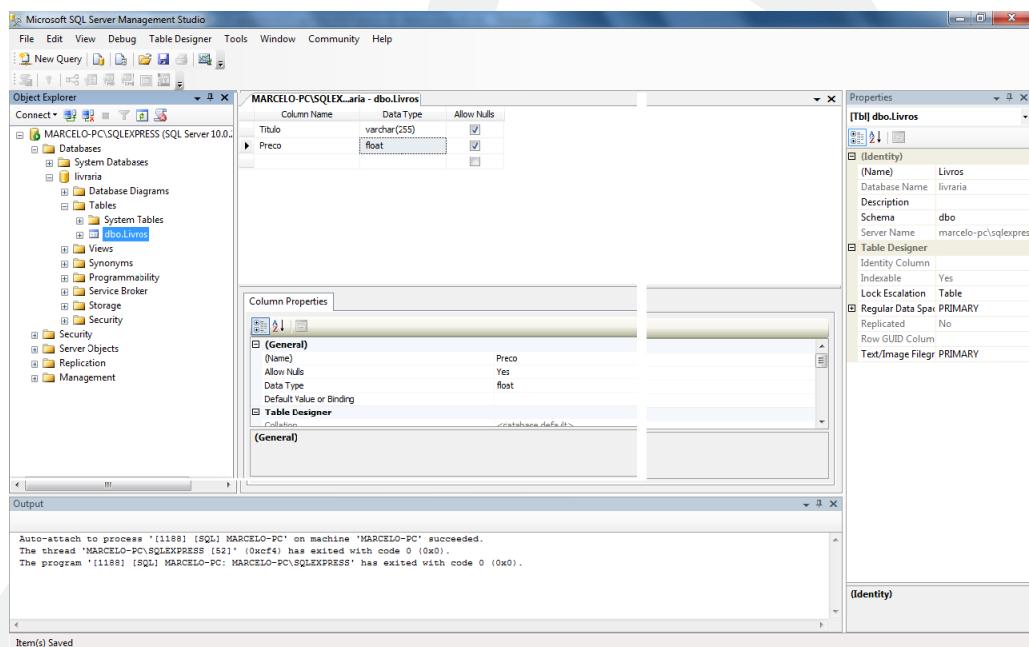
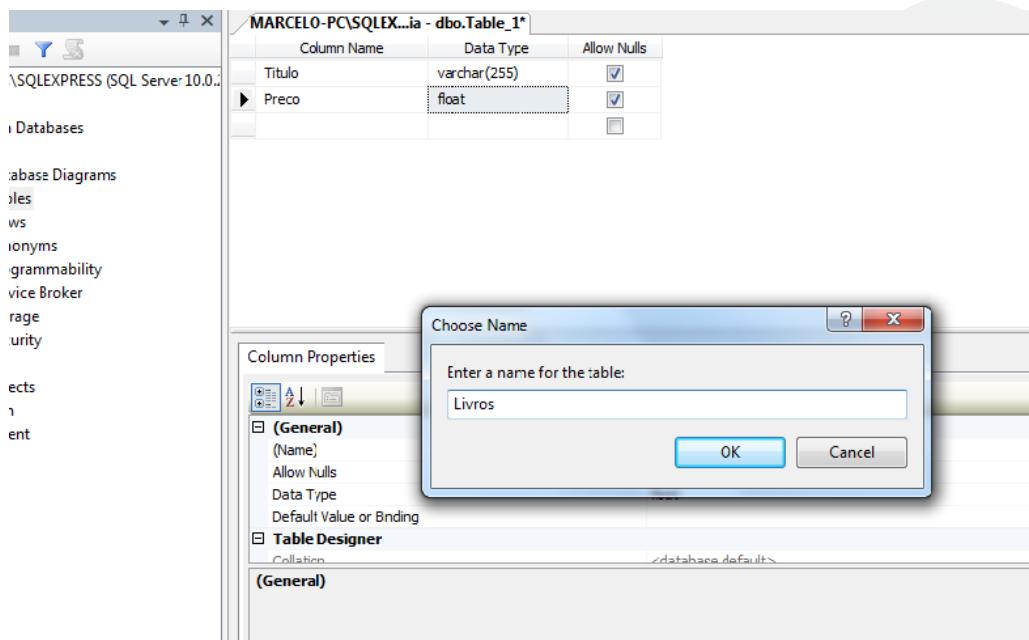
As tabelas no SQL Server Express são criadas através do comando **CREATE TABLE**. Na criação de uma tabela é necessário definir quais são os nomes e os tipos das colunas.



The screenshot shows the SSMS interface. In the Object Explorer, a database named 'livraria' is selected under 'Databases'. In the main pane, the 'Table Designer' is open for 'dbo.Table_1'. The 'Preco' column is currently selected in the grid. The 'Column Properties' pane shows the following details for the 'Preco' column:

Name	Type	Allow Nulls
Preco	float	Yes

This screenshot is identical to the one above, showing the SSMS interface with the 'Table Designer' for 'dbo.Table_1'. However, a red circle has been drawn around the 'Save Table_1' button in the toolbar, which is located between the 'New Query' and 'File' buttons.



No SQL Server os nomes das tabelas são precedidas pelo ID do usuário que possui a tabela. No caso do usuário **sa**, o ID é **dbo**. Portanto o nome da tabela Livros fica ***dbo.Livros***.

Se uma tabela não for mais desejada ela pode ser removida através do comando **DROP TABLE**.

CRUD

As operações básicas para manipular os dados persistidos são: inserir, ler, alterar e remover.

Essas operações são realizadas através de uma linguagem de consulta denominada **SQL (Structured Query Language)**. Essa linguagem oferece quatro comandos básicos: **INSERT, SELECT, UPDATE**

e **DELETE**. Esses comandos são utilizados para inserir, ler, alterar e remover registros, respectivamente.

The screenshot shows two separate sessions in SQL Server Management Studio:

SQLQuery6.sql - MA...livraria (sa (52))*

```
INSERT INTO livraria.dbo.Livros (Titulo,Preco) VALUES ('C#', 98.75);
```

SQLQuery7.sql - MA...livraria (sa (52))*

```
SELECT * FROM livraria.dbo.Livros;
```

Results tab output:

Titulo	Preco
C#	98.75

Messages tab output:

Query executed successfully.

The screenshot shows the SQL Server Management Studio interface. A query window titled "SQLQuery7.sql - MA...livraria (sa (52))" contains the following SQL code:

```
UPDATE livraria.dbo.Livros SET Preco = 115.9 WHERE Titulo = 'C#' ;
```

The results pane below shows the affected row:

	Titulo	Preco
1	C#	115,9

The status bar at the bottom indicates "Connected. (1/1)" and "MARCELO-PC\SQLEXPRESS (10.0... | sa (52) | livraria | 00:00:00 | 0 ro".

The screenshot shows the SQL Server Management Studio interface. A query window titled "SQLQuery7.sql - MA...livraria (sa (52))" contains the following SQL code:

```
SELECT * FROM livraria.dbo.Livros;
```

The results pane below displays the data from the Livros table:

	Titulo	Preco
1	C#	115,9

The status bar at the bottom indicates "Query executed successfully." and "MARCELO-PC\SQLEXPRESS (10.0... | sa (52) | livraria | 00:00:00 | 1 ro".

The screenshot displays two instances of SQL Server Management Studio (SSMS) running side-by-side.

Top Window:

- Query Editor: SQLQuery9.sql - M....master (sa (54))*
- Text: `DELETE FROM livraria.dbo.Livros WHERE Titulo = 'C#';`
- Status Bar: Connected. (1/1) | MARCELO-PC\SQLEXPRESS (10.0... | sa (54) | master | 00:00:00 | 0 rows

Bottom Window:

- Query Editor: SQLQuery9.sql - M....master (sa (54))*
- Text: `SELECT * FROM livraria.dbo.Livros;`
- Status Bar: Connected. (1/1) | MARCELO-PC\SQLEXPRESS (10.0... | sa (54) | master | 00:00:00 | 0 rows
- Results Tab: Shows a table with columns Titulo and Preco, but no data rows.
- Status Bar: Query executed successfully. | MARCELO-PC\SQLEXPRESS (10.0... | sa (54) | master | 00:00:00 | 0 rows

Chaves Primária e Estrangeira

Suponha que os livros da nossa livraria sejam classificados por editoras. As editoras possuem nome e telefone. Para armazenar esses dados, uma nova tabela deveria ser criada.

Nesse momento, teríamos duas tabelas (Livro e Editora). Constantemente, a aplicação da livraria deverá descobrir qual é a editora de um determinado livro ou quais são os livros de uma determinada editora. Para isso, os registros da tabela Editora devem estar relacionados aos da tabela Livro.

Na tabela Livro, poderíamos adicionar uma coluna para armazenar o nome da editora dos livros. Dessa forma, se alguém quiser recuperar as informações da editora de um determinado livro, deve

consultar a tabela Livro para obter o nome da editora correspondente. Depois, com esse nome, deve consultar a tabela Editora para obter as informações da editora.

Porém, há um problema nessa abordagem. A tabela Editora aceita duas editoras com o mesmo nome. Dessa forma, eventualmente, não conseguiríamos descobrir os dados corretos da editora de um determinado livro. Para resolver esse problema, deveríamos criar uma restrição na tabela Editora que proíba a inserção de editoras com o mesmo nome.

Para resolver esse problema no SQL Server Express, poderíamos adicionar a propriedade **UNIQUE** no campo nome da tabela Editora. Porém, ainda teríamos mais um problema. Na tabela Livro, poderíamos adicionar registros vinculados a editoras inexistentes, pois não há nenhuma relação explícita entre as tabelas. Para solucionar esses problemas, devemos utilizar o conceito de **chave primária e chave estrangeira**.

Toda tabela pode ter uma chave primária, que é um conjunto de um ou mais campos que devem ser únicos para cada registro. Normalmente, um campo numérico é escolhido para ser a chave primária de uma tabela, pois as consultas podem ser realizadas com melhor desempenho.

Então, poderíamos adicionar um campo numérico na tabela Editora e torná-lo chave primária. Vamos chamar esse campo de **id**. Na tabela Livro, podemos adicionar um campo numérico chamado **editora_id** que deve ser utilizado para guardar o valor da chave primária da editora correspondente ao livro. Além disso, o campo editora_id deve estar explicitamente vinculado com o campo id da tabela Editora. Para estabelecer esse vínculo, o campo editora_id da tabela Livro deve ser uma chave estrangeira associada à chave primária da tabela Editora.

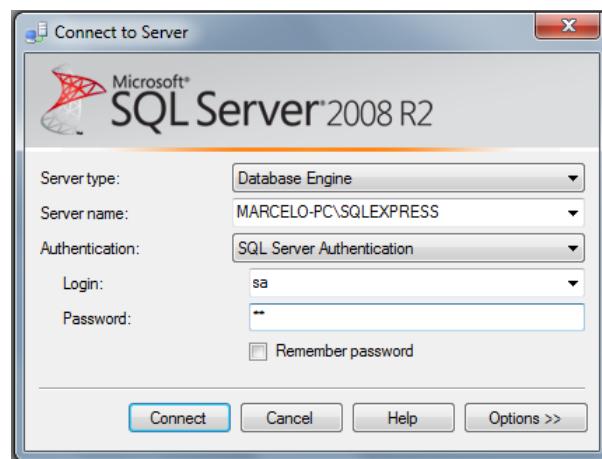
Uma chave estrangeira é um conjunto de uma ou mais colunas de uma tabela que possuem valores iguais aos da chave primária de outra tabela.

Com a definição da chave estrangeira, um livro não pode ser inserido com o valor do campo editora_id inválido. Caso tentássemos fazer isso, obteríamos uma mensagem de erro.

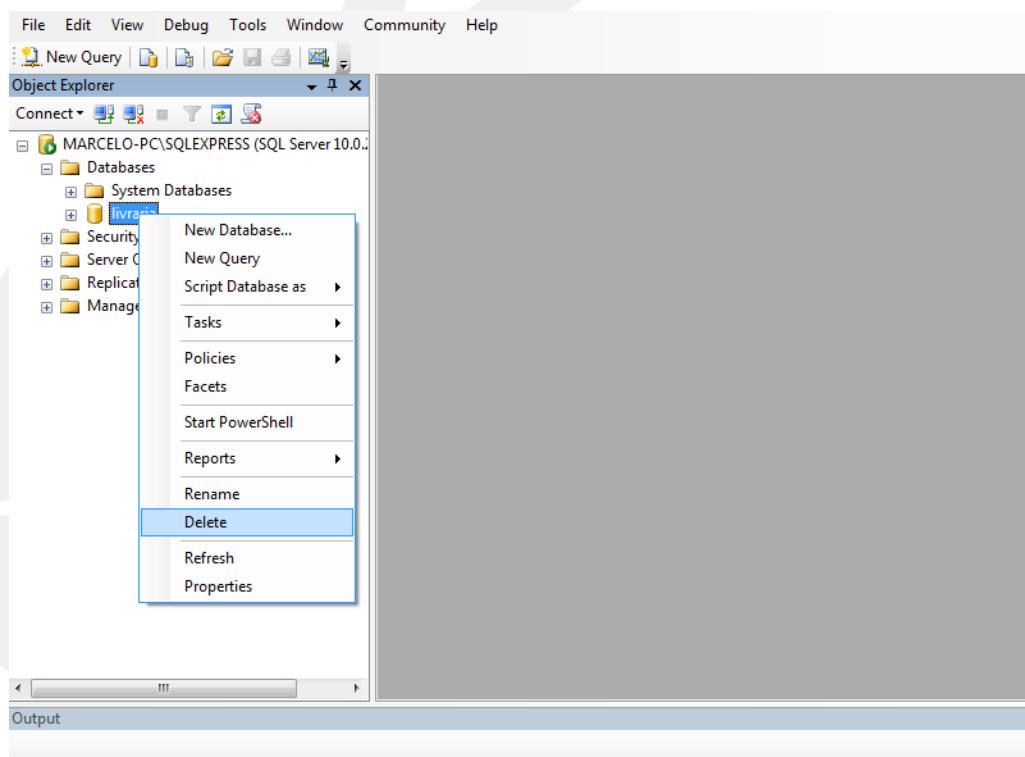


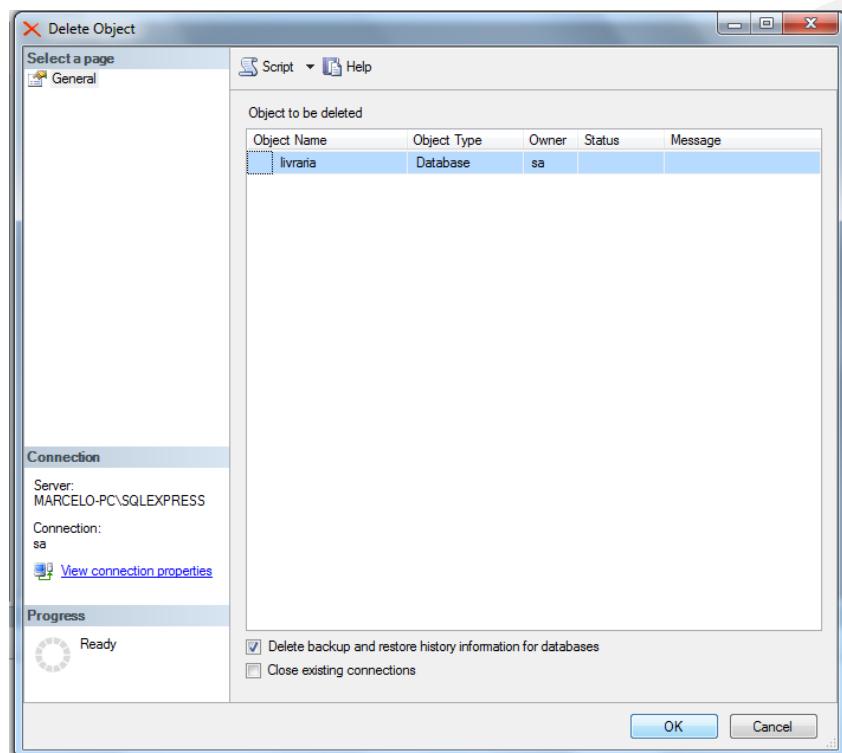
Exercícios de Fixação

- 1 Abra o Microsoft SQL Server Management Studio Express utilizando **NOME_DA_MAQUINA SQLEXPRESS** como Server Name, **SQL Server Authentication** como Authentication, **sa** como Login e **sa** como Password.

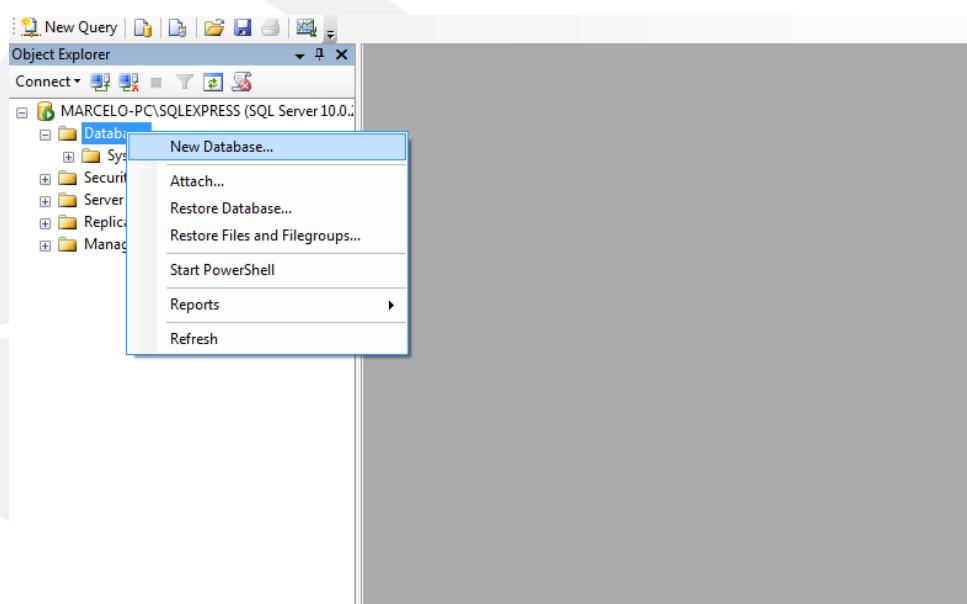


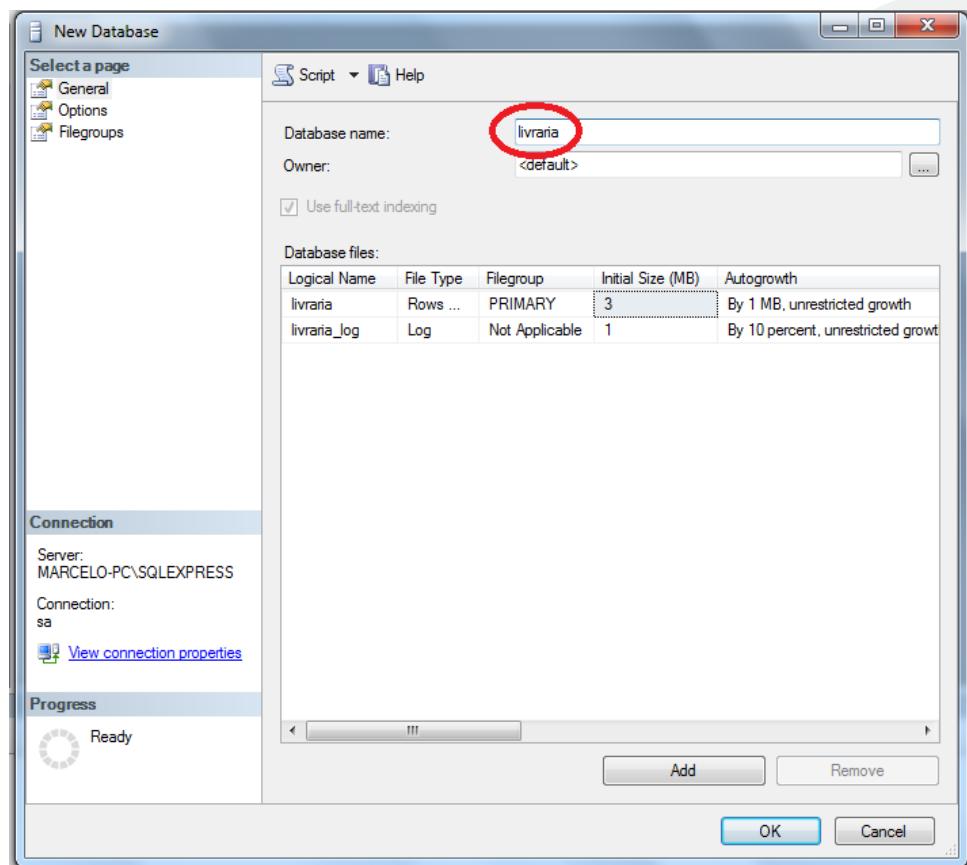
- 2 Caso exista uma base de dados chamada **Livraria**, remova-a conforme a figura abaixo:



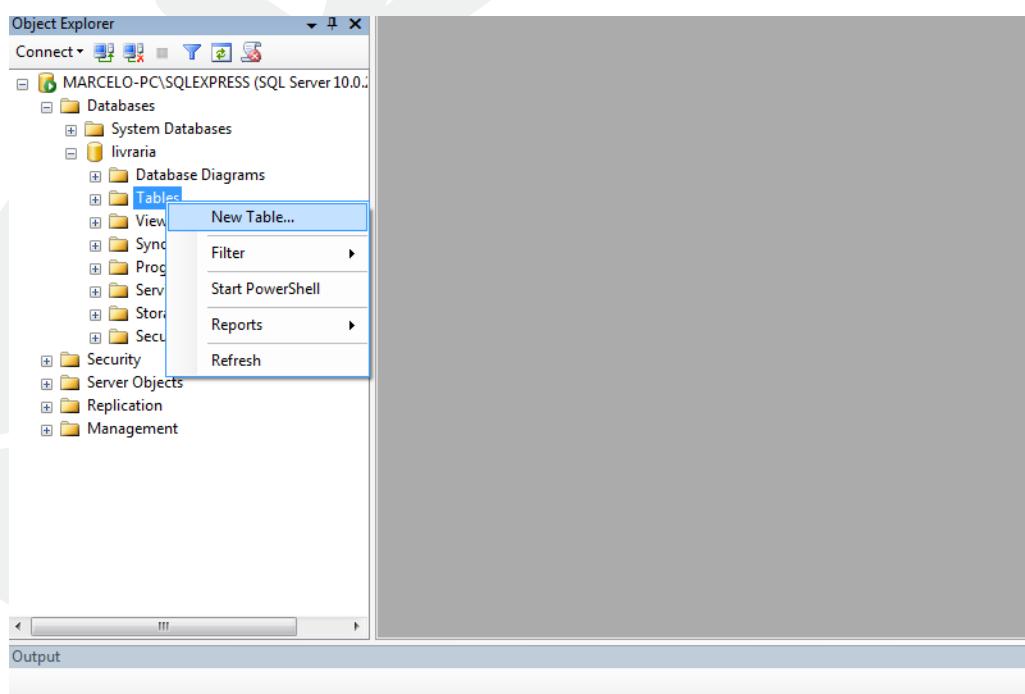


- 3 Crie uma nova base de dados chamada **livraria**, conforme mostrado na figura abaixo. Você vai utilizar esta base nos exercícios seguintes.





- 4 Crie uma tabela chamada **Editoras** conforme as figuras abaixo.



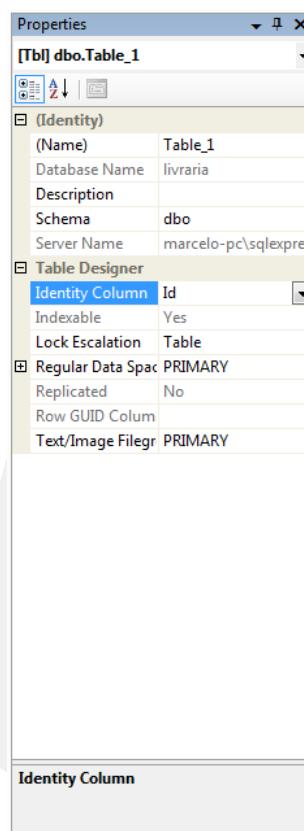
Altere os campos para torná-los obrigatórios, NÃO permitindo que eles fiquem em branco *NULL*.

The screenshot shows the SSMS Table Designer for 'dbo.Table_1'. The 'Column Name' table has three rows: 'Id' (bigint), 'Nome' (varchar(255)), and 'Email' (varchar(255)). The 'Allow Nulls' column for all three rows is checked (set to 'Yes'). A red box highlights the 'Allow Nulls' column header and the first row. Below the table, the 'Column Properties' pane shows the 'General' properties for the 'Id' column, where 'Allow Nulls' is set to 'No'.

Além disso o campo **Id** deve ser uma chave primária.

The screenshot shows the SSMS Table Designer for 'dbo.Table_1'. A context menu is open over the 'Id' column, with the 'Set Primary Key' option highlighted. Other options in the menu include 'Insert Column', 'Delete Column', 'Relationships...', 'Indexes/Keys...', 'Fulltext Index...', 'XML Indexes...', 'Check Constraints...', 'Spatial Indexes...', and 'Generate Change Script...'. The 'Column Name' table shows the 'Id' column as the primary key. The 'Column Properties' pane below shows the 'General' properties for the 'Id' column, where 'Allow Nulls' is set to 'No'.

O campo **Id** dever ser incrementado automaticamente. Defina ele com a propriedade **Identity** segundo a figura abaixo:

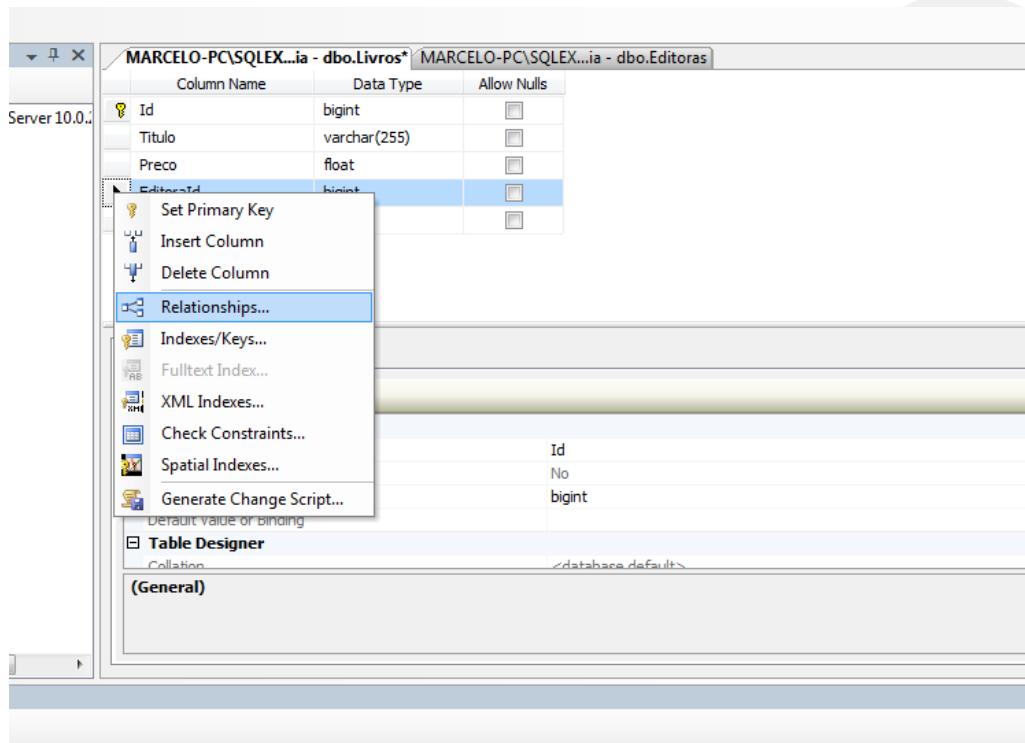


- 5 Crie uma tabela chamada **Livros** conforme as figuras abaixo:

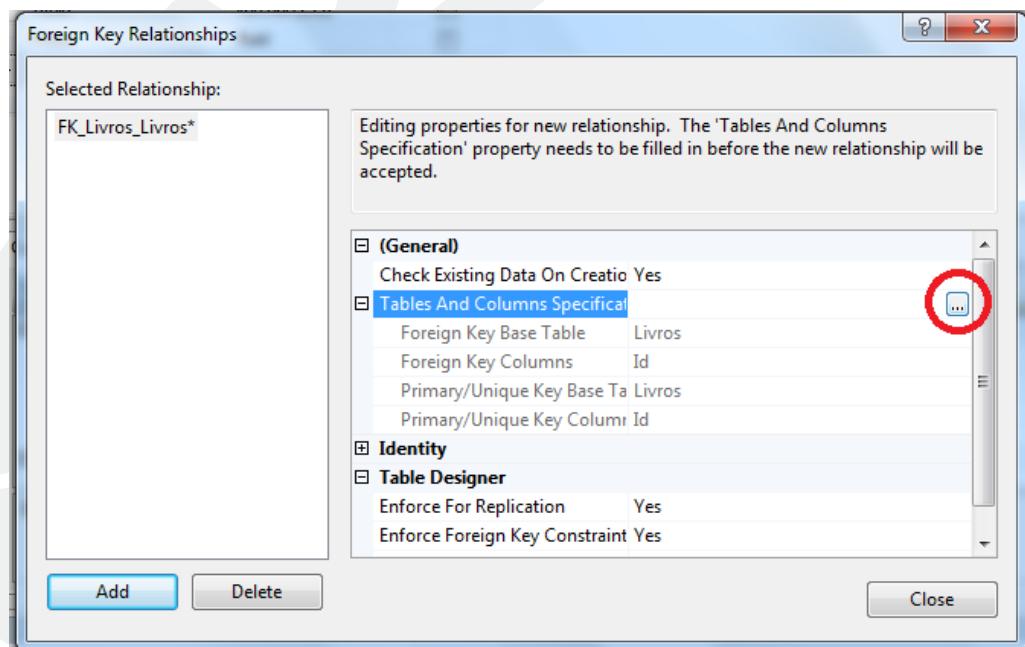
The screenshot shows the SSMS Table Designer interface. At the top, it displays the connection information: 'MARCELO-PC\SQLEX...ia - dbo.Table_1*' and 'MARCELO-PC\SQLEX...ia - dbo.Editors'. Below this is a table structure with four columns: 'Id', 'Titulo', 'Preco', and 'EditoraId'. The 'EditoraId' column is currently selected, indicated by a blue selection box around its row. In the 'Column Properties' pane on the right, under the '(General)' section, the 'Name' is set to 'EditoraId', 'Allow Nulls' is checked (indicated by a grey box), 'Data Type' is 'bigint', and 'Default Value or Binding' is empty. Under the 'Table Designer' section, the 'Collation' is set to 'database_default'. There is also a '(General)' section at the bottom of the properties pane.

Lembrando de NÃO marcar a opção **ALLOW NULL**. Além disso o campo **Id** deve ser uma chave primária e automaticamente incrementada.

Você precisa tornar o campo **EditoraId** uma chave estrangeira. Clique com o botão direito sobre a coluna **EditoraId** e selecione a opção **Relationships...**, conforme a figura abaixo:

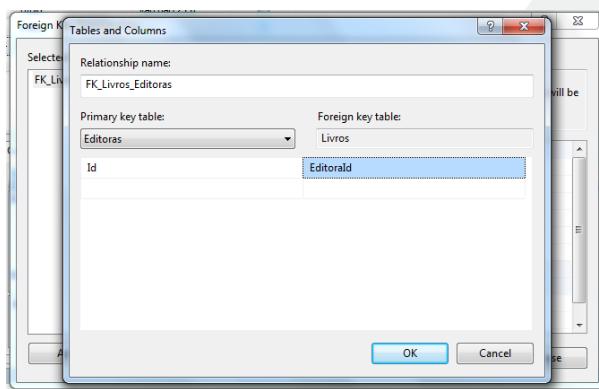


Devemos acrescentar o relacionamento entre livro e editora. Clique em **Add** e posteriormente no botão à direita na linha *Tables and Columns Specification*.



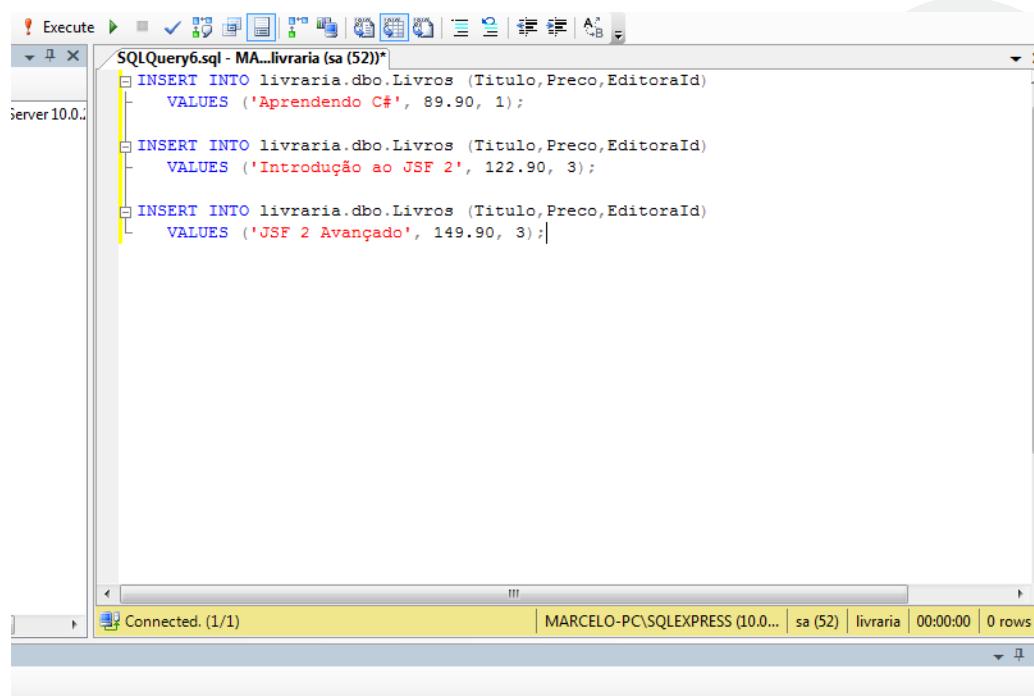
Devemos informar qual é a chave primária que a coluna *EditorialId* da tabela Livros faz referência.

Para isto, informe a tabela Editoras como **Primary Key Table** e indique a coluna *Id* como a chave primária referenciada. Selecione a coluna **EditorId** como a coluna que irá fazer referência a chave primária da tabela Editoras.



- 6 Adicione alguns registros na tabela Editoras. Veja exemplos na figura abaixo:

Adicione alguns registros na tabela Livros. Veja exemplos na figura abaixo:

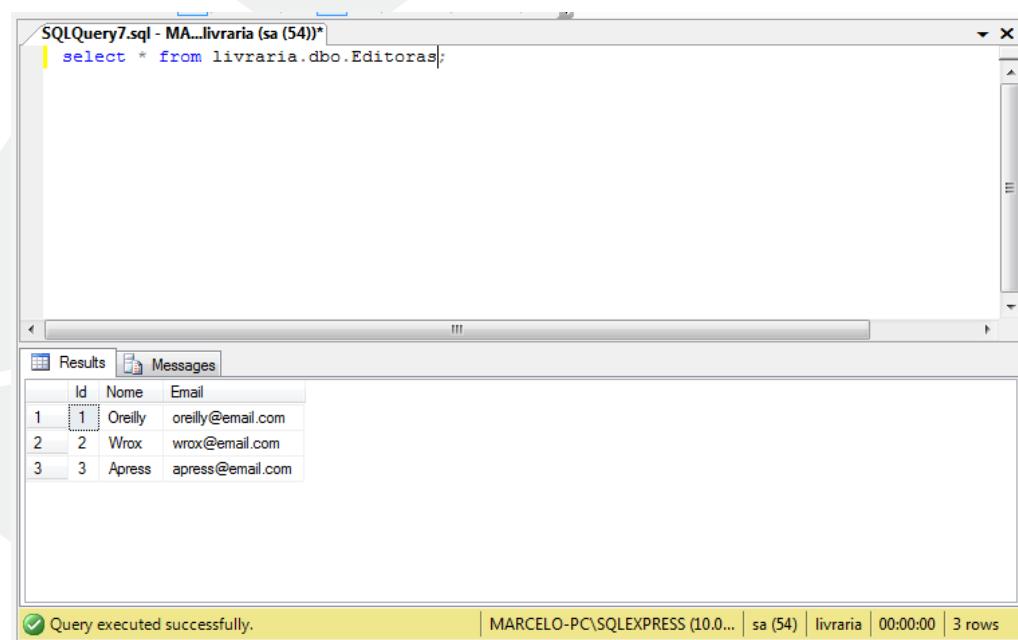


```
SQLQuery6.sql - MA...livraria (sa (52))*
INSERT INTO livraria.dbo.Livros (Titulo,Preco,EditoraId)
VALUES ('Aprendendo C#', 89.90, 1);

INSERT INTO livraria.dbo.Livros (Titulo,Preco,EditoraId)
VALUES ('Introdução ao JSF 2', 122.90, 3);

INSERT INTO livraria.dbo.Livros (Titulo,Preco,EditoraId)
VALUES ('JSF 2 Avançado', 149.90, 3);
```

- 7 Consulte os registros da tabela Editoras, e em seguida consulte a tabela Livros. Veja exemplos logo abaixo:



```
SQLQuery7.sql - MA...livraria (sa (54))*
select * from livraria.dbo.Editoras;
```

	Id	Nome	Email
1	1	Oreilly	oreilly@email.com
2	2	Wrox	wrox@email.com
3	3	Apress	apress@email.com

Query executed successfully.

The screenshot shows a SQL Server Management Studio window. The query editor tab is titled "SQLQuery7.sql - MA...livraria (sa (54))". The query is:

```
select * from livraria.dbo.Livros;
```

The results pane shows a table with four columns: Id, Titulo, Preco, and EditorialId. The data is:

	Id	Titulo	Preco	EditorialId
1	1	Aprendendo C#	89,9	1
2	2	Introdução ao JSF 2	122,9	3
3	3	JSF 2 Avançado	149,9	3

At the bottom of the results pane, a message says "Query executed successfully." and provides connection information: MARCELO-PC\SQLEXPRESS (10.0...) | sa (54) | livraria | 00:00:00 | 3 rows.

- 8 Altere alguns dos registros da tabela Livros. Veja o exemplo abaixo:

The screenshot shows a SQL Server Management Studio window. The query editor tab is titled "SQLQuery9.sql - MA...livraria (sa (52))". The query is:

```
UPDATE livraria.dbo.Livros SET Preco = 92,9 WHERE Id = 1;
```

The results pane shows a message "Connected. (1/1)" at the top. At the bottom, it shows connection information: MARCELO-PC\SQLEXPRESS (10.0...) | sa (52) | livraria | 00:00:00 | 0 rows.

- 9 Altere alguns dos registros da tabela Editoras. Veja o exemplo abaixo:

The screenshot shows a SQL Server Management Studio window titled "SQLQuery10.sql - M...livraria (sa (52))". The query entered is:

```
UPDATE livraria.dbo.Editoras SET Nome = 'O'Reilly' WHERE Id = 1;
```

The status bar at the bottom indicates "Connected. (1/1)" and "MARCELO-PC\SQLEXPRESS (10.0... | sa (52) | livraria | 00:00:00 | 0 rows".

- 10 Remova alguns registros da tabela Livros. Veja o exemplo abaixo:

The screenshot shows a SQL Server Management Studio window titled "SQLQuery11.sql - M...livraria (sa (52))". The query entered is:

```
DELETE FROM livraria.dbo.Livros WHERE Id = 2;
```

The status bar at the bottom indicates "Connected. (1/1)" and "MARCELO-PC\SQLEXPRESS (10.0... | sa (52) | livraria | 00:00:00 | 0 rows".

- 11 Remova alguns registros da tabela Editoras. Preste atenção para não remover uma editora que tenha algum livro relacionado já adicionado no banco. Veja o exemplo abaixo:

The screenshot shows a SQL Server Management Studio window. The query editor tab is titled "SQLQuery12.sql - M...livraria (sa (52))". The query itself is:

```
DELETE FROM livraria.dbo.Editoras WHERE Id = 2;
```

The status bar at the bottom indicates the connection is "Connected. (1/1)" and the session details are "MARCELO-PC\SQLEXPRESS (10.0... | sa (52) | livraria | 00:00:00 | 0 rows".

- 12 Faça uma consulta para buscar todos os livros de uma determinada editora. Veja um exemplo na figura abaixo:

The screenshot shows a SQL Server Management Studio window. The query editor tab is titled "SQLQuery12.sql - M...livraria (sa (52))". The query is:

```
SELECT * FROM livraria.dbo.Livros as L,livraria.dbo.Editoras as E  
WHERE L.EditoraId = E.Id;
```

The results pane shows a table with two rows:

	Id	Titulo	Preco	EditoraId	Id	Nome	Email
1	1	Aprendendo C#	92.9	1	1	O'Reilly	oreilly@email.com
2	3	JSF 2 Avançado	149.9	3	3	Apress	apress@email.com

The status bar at the bottom indicates the query was "Query executed successfully." and the session details are "MARCELO-PC\SQLEXPRESS (10.0... | sa (52) | livraria | 00:00:00 | 2 rows".

ADO.NET

No capítulo anterior, aprendemos que utilizar bancos de dados é uma ótima alternativa para armazenar os dados de uma aplicação. Entretanto, você deve ter percebido que as interfaces disponíveis para interagir com o SQL Server Express não podem ser utilizadas por qualquer pessoa. Para utilizá-las, é necessário conhecer a linguagem SQL e os conceitos do modelo relacional. Em geral, as interfaces dos outros SGDBs exigem os mesmos conhecimentos.

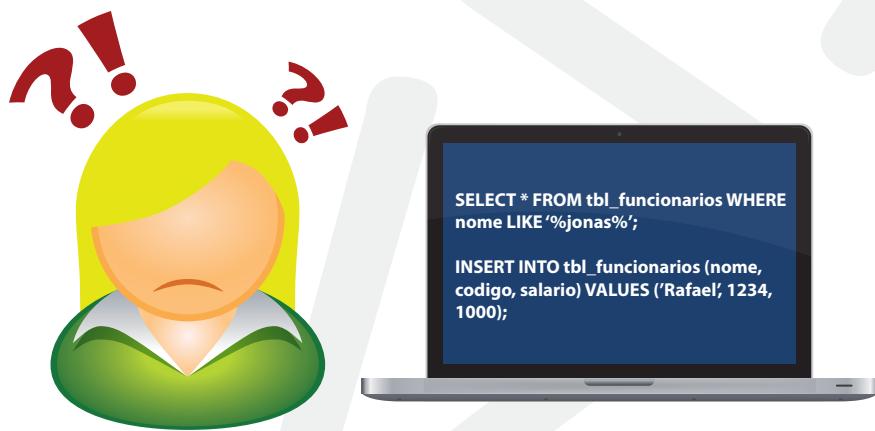


Figura 2.1: Usuários comuns não possuem conhecimento sobre SQL ou sobre o modelo relacional

Para resolver esse problema, podemos desenvolver aplicações com interfaces que não exijam conhecimentos técnicos de SQL ou do modelo relacional para serem utilizadas. Dessa forma, usuários comuns poderiam manipular as informações do banco de dados através dessas aplicações. Nessa abordagem, os usuários interagem com as aplicações e as aplicações interagem com os SGDBs.



Figura 2.2: Usuários comuns devem utilizar interfaces simples

Driver

As aplicações interagem com os SGDBs através de troca de mensagens. Os SGDBs definem o formato das mensagens. Para não sobrecarregar o canal de comunicação entre as aplicações e os SGDBs, as mensagens trocadas devem ocupar o menor espaço possível. Geralmente, protocolos binários são mais apropriados para reduzir o tamanho das mensagens e consequentemente diminuir a carga do canal de comunicação. Por isso, os SGDBs utilizam protocolos binários.

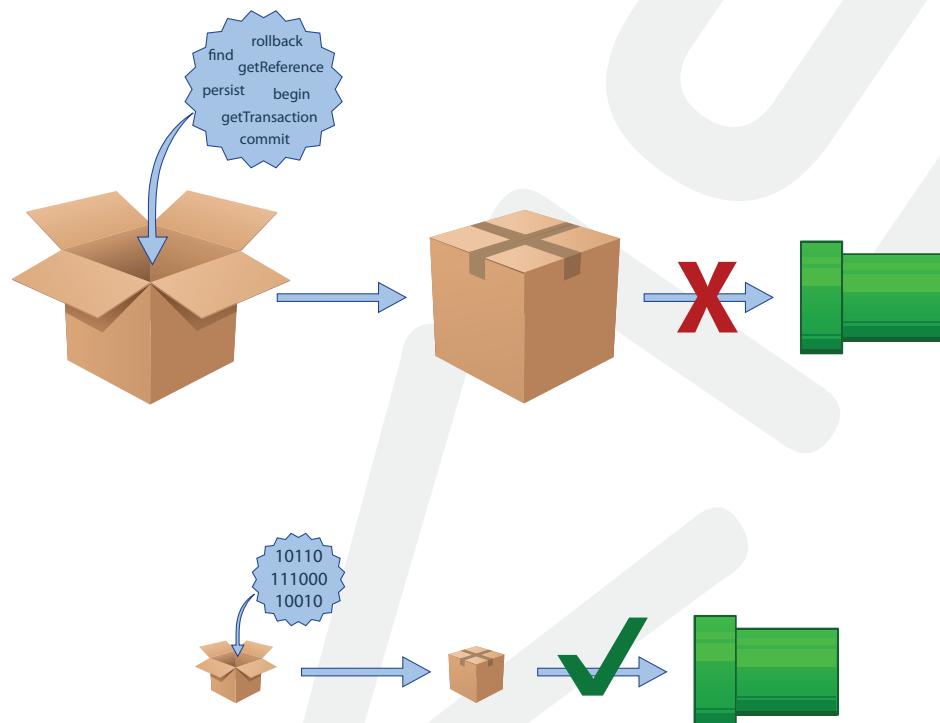


Figura 2.3: Diminuindo o tamanho das mensagens para não sobrecarregar o meio de comunicação

Mensagens binárias são facilmente interpretadas por computadores. Por outro lado, são complexas para um ser humano compreender. Dessa forma, o trabalho dos desenvolvedores seria muito complexo, aumentando o custo para o desenvolvimento e manutenção das aplicações.

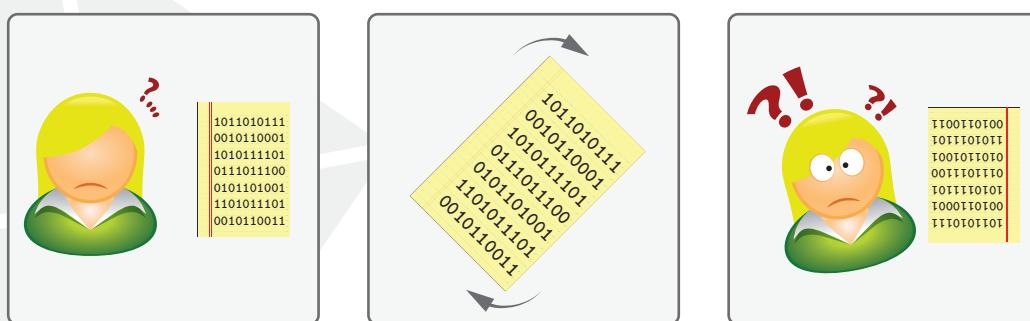


Figura 2.4: Mensagens binárias são altamente complexas para os seres humanos

Para resolver esse problema e facilitar o desenvolvimento das aplicações, as empresas proprietá-

rias dos SGDBs, normalmente, desenvolvem e distribuem **drivers de conexão**. Um driver de conexão atua como um intermediário entre as aplicações e os SGDBs.

Os drivers de conexão são “tradutores” de comandos escritos em uma determinada linguagem de programação para comandos definidos de acordo com o protocolo de um SGDB. Utilizando um driver de conexão, os desenvolvedores das aplicações não manipulam diretamente as mensagens binárias trocadas entre as aplicações e os SGDBs.



Mais Sobre

Em alguns casos, o protocolo binário de um determinado SGDB é fechado. Consequentemente, a única maneira de se comunicar com ele é através de um driver de conexão oferecido pelo fabricante desse SGDB.

ODBC

Suponha que os proprietários dos bancos de dados desenvolvessem os drivers de maneira totalmente independente. Consequentemente, cada driver teria sua própria interface, ou seja, seu próprio conjunto de instruções. Dessa maneira, o desenvolvedor da aplicação precisa conhecer as instruções de cada um dos drivers dos respectivos bancos que ele for utilizar.

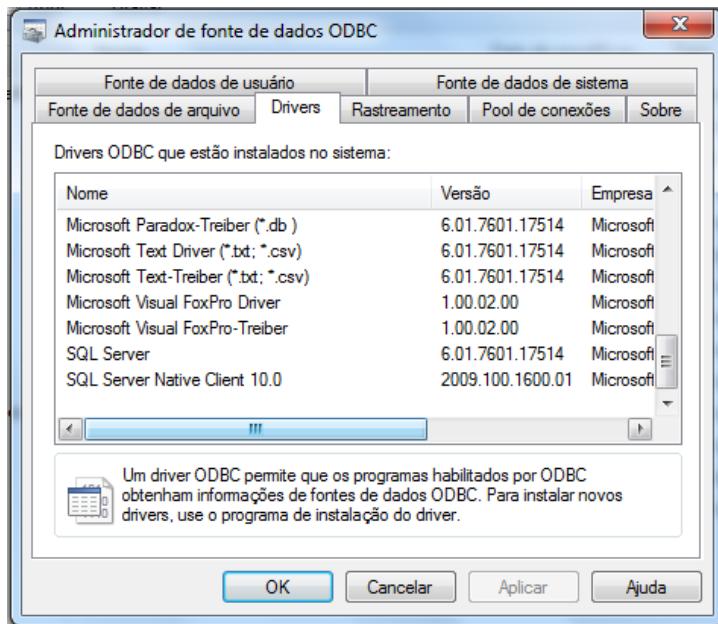
Para facilitar o trabalho do desenvolvedor da aplicação, a Microsoft® definiu uma especificação chamada ODBC (Open Database Connectivity) para padronizar a interface dos drivers de conexão. Assim, quando uma empresa proprietária de um banco de dados pretende desenvolver um driver, ela segue essa especificação com o intuito de popularizá-lo.

Os drivers de conexão que respeitam a especificação ODBC, ou seja, possuem um conjunto de comandos padronizados, são chamados de drivers de conexão ODBC.

ODBC Manager

Para que drivers ODBC possam ser instalados em uma máquina e as aplicações consigam utilizá-los é necessário ter o ODBC Manager, que já vem instalado no Windows®.

O driver de conexão ODBC já está disponível para utilização, podemos consultar o ODBC Manager do Windows®. O ODBC Manager pode ser executado através do item Ferramentas Administrativas do Painel de Controle.



Criando uma conexão

Com o driver de conexão ODBC instalado na máquina já é possível criar uma conexão com o banco de dados correspondente. O que é necessário para estabelecer uma conexão com o banco de dados?

- Escolher o driver de conexão;
- Definir a localização do banco de dados;
- Informar o nome da base de dados;
- Ter um usuário e senha cadastrados no banco de dados.

Todas essas informações são definidas na chamada **string de conexão**.

```
1 string stringDeConexao = @"driver={SQL Server};  
2   server=MARCELO-PC\SQLEXPRESS;database=livraria;uid=sa;pwd=sa;"
```

Código C# 2.1: Definindo a string de conexão

Após a definição da string de conexão, podemos utilizar a classe **System.Data.Odbc.OdbcConnection** do *.NET Framework*. Essa classe é responsável por criar conexões ODBC.

```
1 OdbcConnection conexao = new OdbcConnection(stringDeConexao);
```

Código C# 2.2: Criando uma conexão ODBC

Inserindo registros

Estabelecida a conexão com o banco de dados, já podemos executar comandos. Por exemplo, é possível inserir registros nas tabelas. O primeiro passo para executar um comando é defini-lo em linguagem SQL de acordo com a sintaxe do SGDB utilizado.

```
1 string textoDoComando = @"INSERT INTO Editoras (Nome, Email)
2   VALUES ('Abril', 'abril@email.com');";
```



Mais Sobre

O caractere “@” antes de um valor literal do tipo string indica que os caracteres dentro da string não devem ser processados como caracteres especiais.

Em seguida, devemos criar um objeto da classe *System.Data.Odbc.OdbcCommand* a partir do código sql e da conexão previamente criados. O comando não é executado quando os objetos dessa classe são instanciados.

```
1 OdbcCommand comando = new OdbcCommand(textoDoComando, conexao);
```

Código C# 2.4: Criando um comando ODBC

Por fim, o comando pode ser executado através do método *ExecuteNonQuery()*. A conexão deve ser aberta antes de executar o comando.

```
1 conexao.Open();
2 comando.ExecuteNonQuery();
```



Importante

A mesma conexão pode ser reutilizada para executar várias operações. Quando não houver mais operações a serem executadas, devemos finalizar a conexão ODBC através do método *Close()*. Finalizar as conexões ODBC que não são mais necessárias é importante pois libera recursos no SGBD.

```
1 conexao.Close();
```

Código C# 2.6: Finalizando uma conexão ODBC



Mais Sobre

Em C#, para evitar que uma conexão não seja fechada após a sua utilização, podemos aplicar um bloco **using**.

```
1 using (OdbcConnection conexao = new OdbcConnection(stringDeConexao))
2 {
3   // utiliza a conexao
4 }
```

Código C# 2.7: Utilizando um bloco using

No código acima, quando o bloco **using** que está associado à conexão ODBC terminar, automaticamente, essa conexão será fechada.



Exercícios de Fixação

- 1 Crie um projeto do tipo Console Application no Microsoft Visual C# Express, chamado **ODBC**.

- 2 Crie uma classe chamada **InsereEditora** no projeto **ODBC** para inserir registros na tabela *Editoras*.

```
1 using System.Data.Odbc;
2
3 namespace Odbc
4 {
5     class InsereEditora
6     {
7         static void Main(string[] args)
8         {
9             string stringDeConexao = @"driver={SQL Server};
10                server=MARCELO-PC\SQLEXPRESS;database=livraria;uid=sa;pwd=sa;";
11
12             System.Console.Write("Digite o Nome da Editora:");
13             string nome = System.Console.ReadLine();
14
15             System.Console.Write("Digite o Email da Editora:");
16             string email = System.Console.ReadLine();
17
18             string textoInsereEditora =
19                 @"INSERT INTO Editoras (Nome, Email)
20                   VALUES('"+ nome + @"', '" + email + @")";
21
22             using (OdbcConnection conexao = new OdbcConnection(stringDeConexao))
23             {
24                 OdbcCommand command = new OdbcCommand(textoInsereEditora, conexao);
25                 conexao.Open();
26                 command.ExecuteNonQuery();
27             }
28         }
29     }
30 }
```

Código C# 2.8: *InsereEditora.cs*



Exercícios Complementares

- 1 Crie uma classe chamada **InsereLivro** no projeto **ODBC** para inserir registros na tabela *Livros*.

SQL Injection

A implementação da inserção de registros feita anteriormente possui uma falha grave. Os dados obtidos do usuário através do teclado não são tratados antes de serem enviados para o SGDB.

Esses dados podem conter caracteres especiais. Se esses caracteres não são tratados, o comportamento esperado da operação é afetado. Eventualmente, registros não são inseridos como deveriam ou brechas de segurança podem se abrir.

Por exemplo, considere a classe **InsereEditora** do exercício de fixação. Se o usuário digitar

“O'Reilly” e “oreilly@email.com”, o código SQL gerado pela aplicação seria:

```
1 INSERT INTO Editoras (nome, email) VALUES ('O'Reilly', 'oreilly@email.com')
```

Observe que o caractere aspas simples aparece cinco vezes no código SQL acima. O SGDB não saberia dizer onde de fato termina o nome da editora. Ao tentar executar esse código, um erro de sintaxe é lançado pelo SQL Server. Para resolver esse problema manualmente, devemos adicionar dois caracteres “” seguidos.

```
1 INSERT INTO Editoras (nome, email) VALUES ('O''Reilly', 'oreilly@email.com')
```

Os valores recebidos dos usuários devem ser analisados e os caracteres especiais contidos nesses valores devem ser tratados. Esse processo é extremamente trabalhoso, pois o conjunto de caracteres especiais e a forma de tratá-los é diferente em cada SGDB.

A responsabilidade do tratamento dos caracteres especiais contidos nos valores de entrada dos usuários pode ser repassada para os drivers ODBC. Dessa forma, o código das aplicações se torna independente das particularidades desse processo para cada SGDB.



Mais Sobre

O processo de tratamento dos caracteres especiais das entradas dos usuários é denominado **sanitize**.

```
1 // pegando os dados da editora pelo teclado
2 string nome = System.Console.ReadLine();
3 string email = System.Console.ReadLine();
4
5 // definindo a sentença SQL com parâmetros
6 string textoDoComando =
7     @"INSERT INTO Editoras (Nome, Email) VALUES (?, ?);";
8
9 // criando um comando odbc
10 OdbcCommand comando = new OdbcCommand(textoDoComando, conexao);
11
12 // atribuindo valores aos parâmetros
13 comando.Parameters.AddWithValue("@Nome", nome);
14 comando.Parameters.AddWithValue("@Email", email);
```

Código C# 2.12: “Sanitizando” as entradas dos usuários

Observe que a sentença SQL foi definida com parâmetros através do caractere “?”. Antes de executar o comando, é necessário atribuir valores aos parâmetros. Isso é feito com o método `AddWithValue()`. Esse método realiza a tarefa de “sanitizar” os valores enviados pelo usuário.



Exercícios de Fixação

- 3 Tente causar um erro de SQL Injection ao inserir editoras com a classe `InsereEditora`. (Dica: tente entradas com aspas simples)

- 4 Altere o código da classe InsereEditora para eliminar o problema do SQL Injection. Observe o código abaixo:

```
1 using System.Data.Odbc;
2
3 namespace Odbc
4 {
5     class InsereEditora
6     {
7         static void Main(string[] args)
8         {
9             string stringDeConexao = @"driver={SQL Server};
10                server=MARCELO-PC\SQLEXPRESS;database=livraria;uid=sa;pwd=sa;";
11
12             System.Console.Write("Digite o Nome da Editora:");
13             string nome = System.Console.ReadLine();
14
15             System.Console.Write("Digite o Email da Editora:");
16             string email = System.Console.ReadLine();
17
18             string textoInsereEditora =
19                 @"INSERT INTO Editoras (Nome, Email) VALUES(?,?)";
20
21             using (OdbcConnection conexao = new OdbcConnection(stringDeConexao))
22             {
23                 OdbcCommand command = new OdbcCommand(textoInsereEditora, conexao);
24
25                 command.Parameters.AddWithValue("@Nome", nome);
26                 command.Parameters.AddWithValue("@Email", email);
27
28                 conexao.Open();
29                 command.ExecuteNonQuery();
30             }
31         }
32     }
33 }
```

Código C# 2.13: InsereEditora.cs

- 5 Agora tente causar novamente o problema de SQL Injection ao inserir novas editoras.



Exercícios Complementares

- 2 Provoque um erro de SQL Injection na classe InsereLivro. (Dica: tente entradas com aspas simples.)
- 3 Altere o código para eliminar o problema do SQL Injection.
- 4 Agora tente causar novamente o problema de SQL Injection ao inserir novos livros.

Listando registros

Depois de inserir alguns registros, é interessante consultar os dados das tabelas para conferir se a inserção foi realizada com sucesso.

O processo para executar um comando de consulta é parecido com o de inserção. É necessário definir a sentença SQL e criar um objeto da classe *OdbcCommand*.

```
1 // definindo a sentença SQL
2 string textoDoComando = @"SELECT * FROM Editoras;";
3
4 // criando um comando odbc
5 OdbcCommand comando = new OdbcCommand(textoDoComando, conexao);
```

Código C# 2.15: Criando um comando de seleção

A diferença é que para executar um comando de consulta é necessário utilizar o método *ExecuteReader()* ao invés do *ExecuteNonQuery()*. Esse método devolve um objeto da classe *System.Data.Odbc.OdbcDataReader*.

```
1 OdbcDataReader resultado = comando.ExecuteReader();
```

Código C# 2.16: Executando um comando de consulta

Os dados contidos no *OdbcDataReader* podem ser acessados através dos nomes das colunas.

```
1 string nome = resultado["Nome"] as string;
2 string email = resultado["Email"] as string;
```

Código C# 2.17: Recuperando os campos do primeiro registro do resultado

O código acima mostra como os campos do primeiro registro do resultado da consulta são recuperados. Agora, para recuperar os outros registros é necessário avançar o *OdbcDataReader* através do método *Read()*.

```
1 string nome1 = resultado["nome"] as string;
2 string email1 = resultado["email"] as string;
3
4 resultado.Read();
5
6 string nome2 = resultado["nome"] as string;
7 string email2 = resultado["email"] as string;
```

Código C# 2.18: Recuperando os campos dos dois primeiros registros do resultado

O próprio método *Read()* devolve um valor booleano para indicar se o reader conseguiu avançar para o próximo registro. Quando esse método devolver *false* significa que não há mais registros para serem recuperados.

```
1 while(resultado.Read())
2 {
3     string nome = resultado["Nome"] as string;
4     string email = resultado["Email"] as string;
5 }
```

Código C# 2.19: Recuperando os campos de todos os registros do resultado



Exercícios de Fixação

- 6 Insira algumas editoras utilizando a classe `InsereEditora` que você criou nos exercícios anteriores.

- 7 Adicione uma nova classe ao projeto **ODBC** chamada `ListaEditora`. O objetivo é listar as editoras que foram salvas no banco. Veja o código dessa classe.

```
1 using System.Data.Odbc;
2
3 namespace Odbc
4 {
5     class ListaEditora
6     {
7         static void Main(string[] args)
8         {
9             string stringDeConexao = @"driver={SQL Server};
10                server=MARCELO-PC\SQLEXPRESS;database=livraria;uid=sa;pwd=sa;";
11
12             using (OdbcConnection conexao = new OdbcConnection(stringDeConexao))
13             {
14                 string textoListaEditora = "SELECT * FROM Editoras";
15                 OdbcCommand command = new OdbcCommand(textoListaEditora, conexao);
16                 conexao.Open();
17                 OdbcDataReader resultado = command.ExecuteReader();
18
19                 while (resultado.Read())
20                 {
21                     long? id = resultado["Id"] as long?;
22                     string nome = resultado["Nome"] as string;
23                     string email = resultado["Email"] as string;
24                     System.Console.WriteLine("{0} : {1} - {2}\n", id, nome, email);
25                 }
26             }
27         }
28     }
29 }
```

Código C# 2.20: `ListaEditora.cs`



Exercícios Complementares

- 5 Crie uma classe para listar os livros cadastrados na base de dados.

Connection Factory

Você deve ter percebido que para cada ação executada no banco de dados, nós precisamos criar uma conexão. Isso gera um problema relacionado à string de conexão ficar armazenada em diversos locais. Imagine que o driver do banco foi atualizado e mudamos a sua versão. Isso implicaria fazer diversas alterações no código em cada ocorrência da string de conexão, tornando o código mais suscetível a erros e dificultando a sua manutenção.

Para resolver esta situação, nós poderíamos criar uma classe responsável pela criação e distribuição de conexões, mantendo assim uma única referência para a string de conexão, e qualquer alteração no modo em que nos conectamos à base de dados, só implica mudanças nesta classe.

```

1 static class ConnectionFactory
2 {
3     public static OdbcConnection CreateConnection()
4     {
5         string driver = @"SQL Server";
6         string servidor = @"MARCELO-PC\SQLEXPRESS";
7         string baseDeDados = @"livraria";
8         string usuario = @"sa";
9         string senha = @"sa";
10
11        StringBuilder connectionString = new StringBuilder();
12        connectionString.Append("driver=");
13        connectionString.Append(driver);
14        connectionString.Append(";server=");
15        connectionString.Append(servidor);
16        connectionString.Append(";database=");
17        connectionString.Append(baseDeDados);
18        connectionString.Append(";uid=");
19        connectionString.Append(usuario);
20        connectionString.Append(";pwd=");
21        connectionString.Append(senha);
22
23        return new OdbcConnection(connectionString.ToString());
24    }
25}

```

Código C# 2.22: ConnectionFactory.cs

Agora podemos obter uma nova conexão apenas chamando `ConnectionFactory.CreateConnection()`. O resto do sistema não precisa mais conhecer os detalhes sobre a conexão com o banco de dados, diminuindo o acoplamento da aplicação.



Exercícios de Fixação

- 8 Adicione uma nova classe chamada `ConnectionFactory` com seguinte código:

```

1 using System;
2 using System.Data.Odbc;
3 using System.Text;
4
5 namespace Odbc
6 {
7     static class ConnectionFactory
8     {
9         public static OdbcConnection CreateConnection()
10        {
11            string driver = @"SQL Server";
12            string servidor = @"MARCELO-PC\SQLEXPRESS";
13            string baseDeDados = @"livraria";
14            string usuario = @"sa";
15            string senha = @"sa";
16
17            StringBuilder connectionString = new StringBuilder();
18            connectionString.Append("driver=");
19            connectionString.Append(driver);
20            connectionString.Append(";server=");
21            connectionString.Append(servidor);
22            connectionString.Append(";database=");
23            connectionString.Append(baseDeDados);
24            connectionString.Append(";uid=");
25            connectionString.Append(usuario);
26            connectionString.Append(";pwd=");

```

```

27     connectionString.Append(senha);
28
29     return new OdbcConnection(connectionString.ToString());
30   }
31 }
32 }
```

Código C# 2.23: ConnectionFactory.cs

- 9** Altere as classes InsereEditora e ListaEditora para que elas utilizem a fábrica de conexão. Execute-as novamente.

```

1  using System.Data.Odbc;
2
3 namespace Odbc
4 {
5   class InsereEditora
6   {
7     static void Main(string[] args)
8     {
9       System.Console.Write("Digite o Nome da Editora:");
10      string nome = System.Console.ReadLine();
11
12      System.Console.Write("Digite o Email da Editora:");
13      string email = System.Console.ReadLine();
14
15      string textoInsereEditora =
16        @"INSERT INTO Editoras (Nome, Email) VALUES(?,?)";
17
18      using (OdbcConnection conexao = ConnectionFactory.CreateConnection())
19      {
20        OdbcCommand command = new OdbcCommand(textoInsereEditora, conexao);
21
22        command.Parameters.AddWithValue("@Nome", nome);
23        command.Parameters.AddWithValue("@Email", email);
24
25        conexao.Open();
26        command.ExecuteNonQuery();
27      }
28    }
29  }
30 }
```

Código C# 2.24: InsereEditora.cs

```

1  using System.Data.Odbc;
2
3 namespace Odbc
4 {
5   class ListaEditora
6   {
7     static void Main(string[] args)
8     {
9       using (OdbcConnection conexao = ConnectionFactory.CreateConnection())
10      {
11        string textoListaEditora = "SELECT * FROM Editoras";
12        OdbcCommand command = new OdbcCommand(textoListaEditora, conexao);
13        conexao.Open();
14        OdbcDataReader resultado = command.ExecuteReader();
15
16        while (resultado.Read())
17        {
18          long? id = resultado["Id"] as long?;
19          string nome = resultado["Nome"] as string;
20          string email = resultado["Email"] as string;
21          System.Console.WriteLine("{0} : {1} - {2}\n", id, nome, email);
22        }
23      }
24    }
25  }
26 }
```

```
22     }
23 }
24 }
25 }
26 }
```

Código C# 2.25: *ListaEditora.cs*



Exercícios Complementares

- 6 Altere as classes `InsereLivro` e `ListaLivro` para que elas utilizem a fábrica de conexão. Execute-as novamente.



Desafios

- 1 Implemente um teste que remove uma editora pelo id.
- 2 Implemente um teste que altera os dados de uma editora pelo id.



ENTITY FRAMEWORK

Múltiplas sintaxes da linguagem SQL

No capítulo anterior, utilizamos conexões ODBC para fazer uma aplicação C# interagir com os SGDBs. Nessa interação, as consultas são definidas com a linguagem SQL. A sintaxe dessa linguagem é diferente em cada SGDB. Dessa forma, a complexidade do trabalho dos desenvolvedores aumenta. Para resolver esse problema, as consultas deveriam ser definidas através de um mecanismo independente da linguagem SQL.

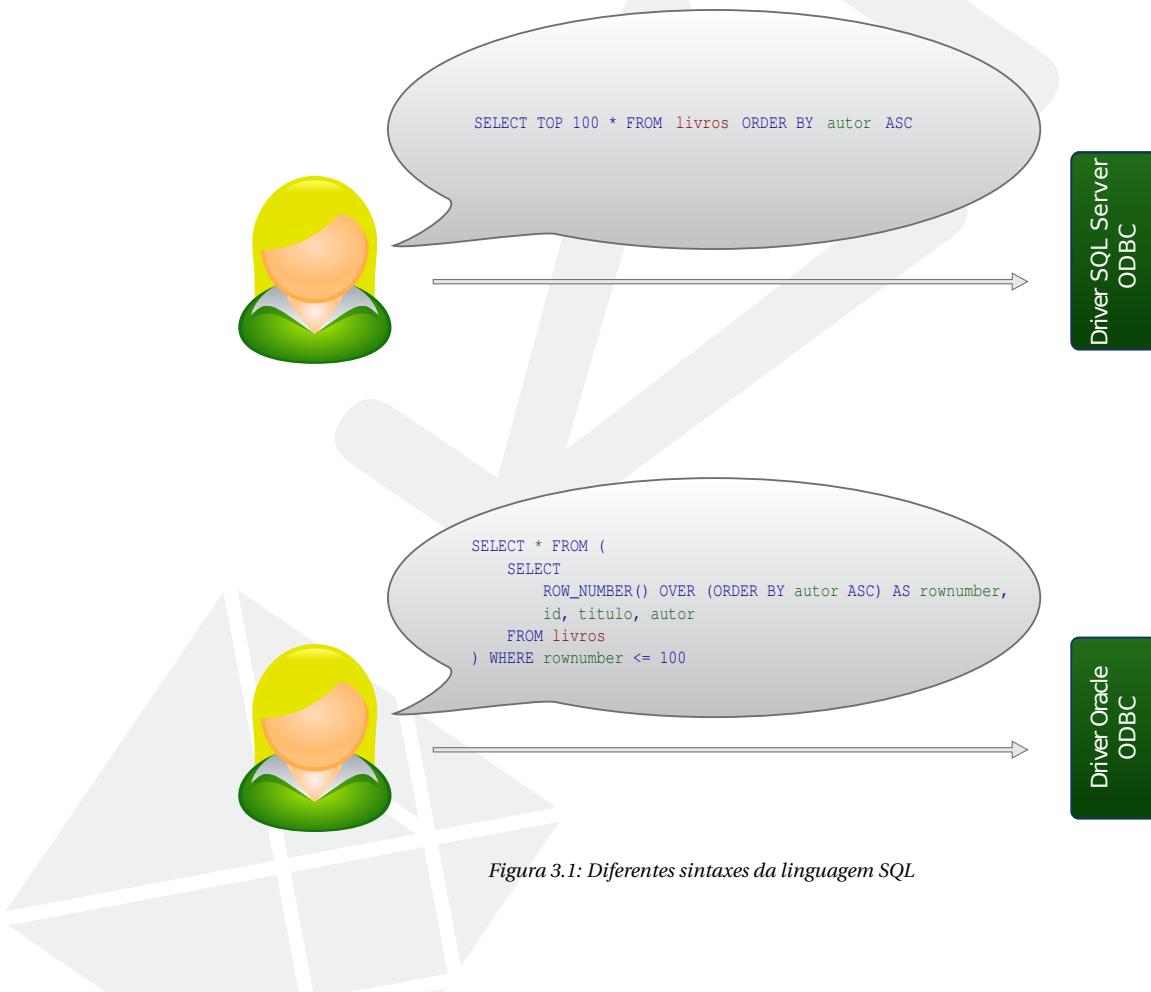


Figura 3.1: Diferentes sintaxes da linguagem SQL

Orientação a Objetos VS Modelo Relacional

Outro problema na comunicação entre uma aplicação C# e um SGDB é o conflito de paradigmas. Os SGDBs são organizados segundo o modelo relacional. Por outro lado, as aplicações C# utilizam

o modelo orientado a objetos.

A transição de dados entre o modelo relacional e o modelo orientado a objetos não é simples. Para realizar essa transição, é necessário definir um mapeamento entre os conceitos desses dois paradigmas. Por exemplo, classes podem ser mapeadas para tabelas, objetos para registros, atributos para campos e referência entre objetos para chaves estrangeiras.

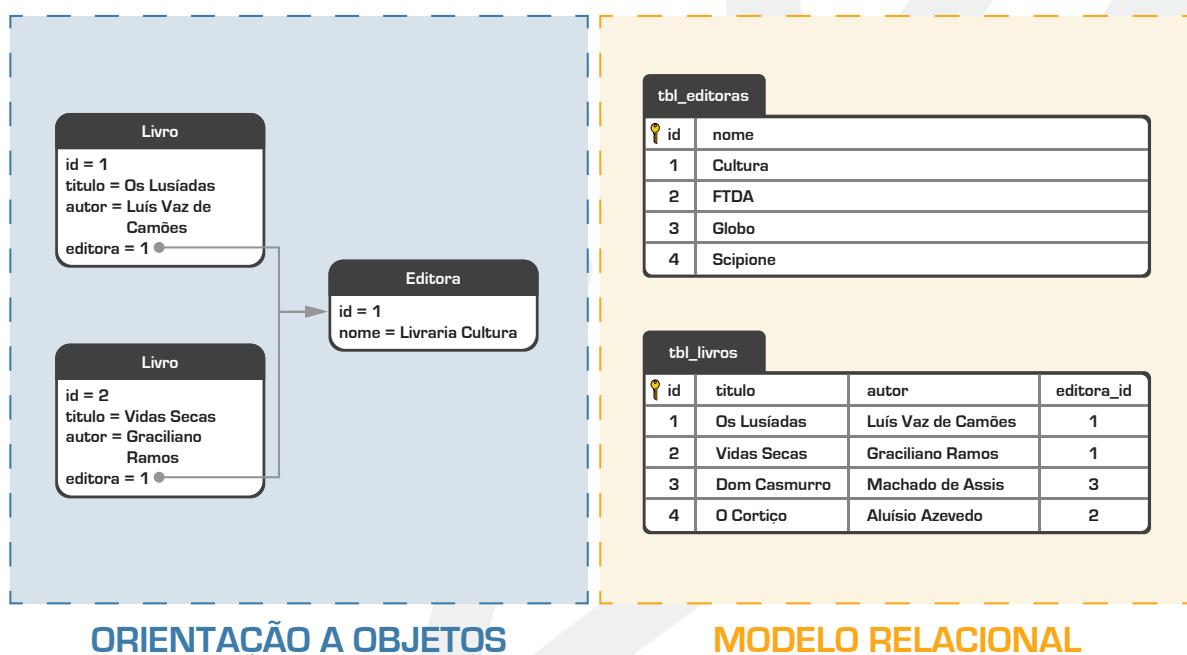


Figura 3.2: Modelo Orientado a Objetos vs Modelo Relacional

Ferramentas ORM

Para facilitar a comunicação entre aplicações C# que seguem o modelo orientado a objetos e os SGDBs que seguem o modelo relacional, podemos utilizar ferramentas que automatizam a transição de dados entre as aplicações e os SGDBs. Essas ferramentas são conhecidas como ferramentas **ORM** (*Object Relational Mapper*).

As ferramentas ORM oferecem mecanismos de consultas independentes da linguagem SQL. Dessa forma, o acoplamento entre as aplicações e os SGDBs diminui drasticamente. As principais ferramentas ORM para C# utilizadas no mercado são o Entity Framework e o NHibernate. Nesse curso, utilizaremos o Entity Framework 5.0.0.



Figura 3.3: Transformação dos dados do Modelo Relacional para o Modelo Orientado a Objetos



Figura 3.4: Transformação dos dados do Modelo Orientado a Objetos para o Modelo Relacional

Instalando o Entity Framework

A instalação do Entity Framework em um projeto do **Visual Studio 2012** pode ser facilmente realizada através do **Package Manager Console**. Veja o comando de instalação a seguir.

```
PM> Install-Package EntityFramework -Version 5.0.0
```

Terminal 3.1: Package Manager Console



Mais Sobre

O Entity Framework também pode ser instalado através do **Manage NuGet Packages**. Basicamente, o Manage NuGet Packages é a alternativa visual ao Package Manager Console.

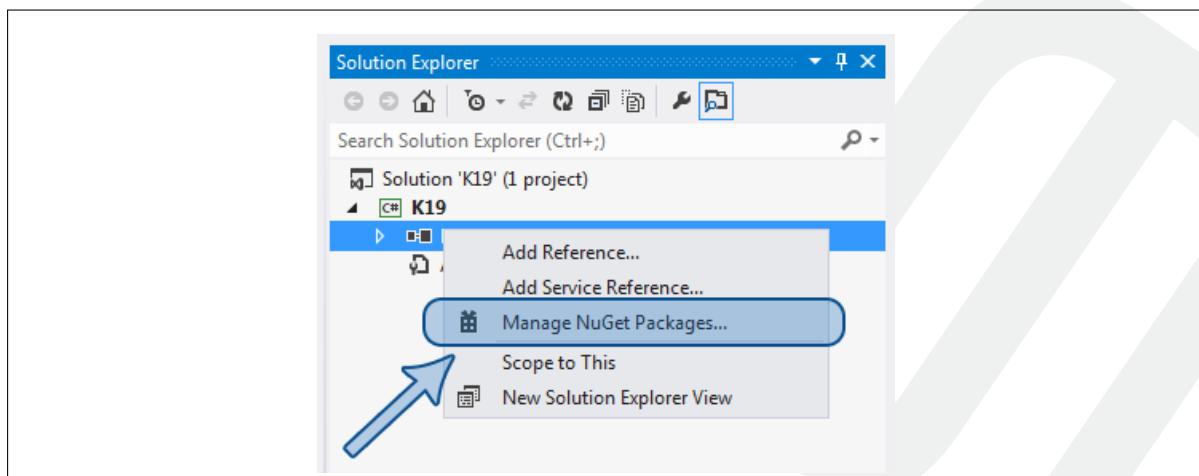


Figura 3.5: Manage NuGet Package

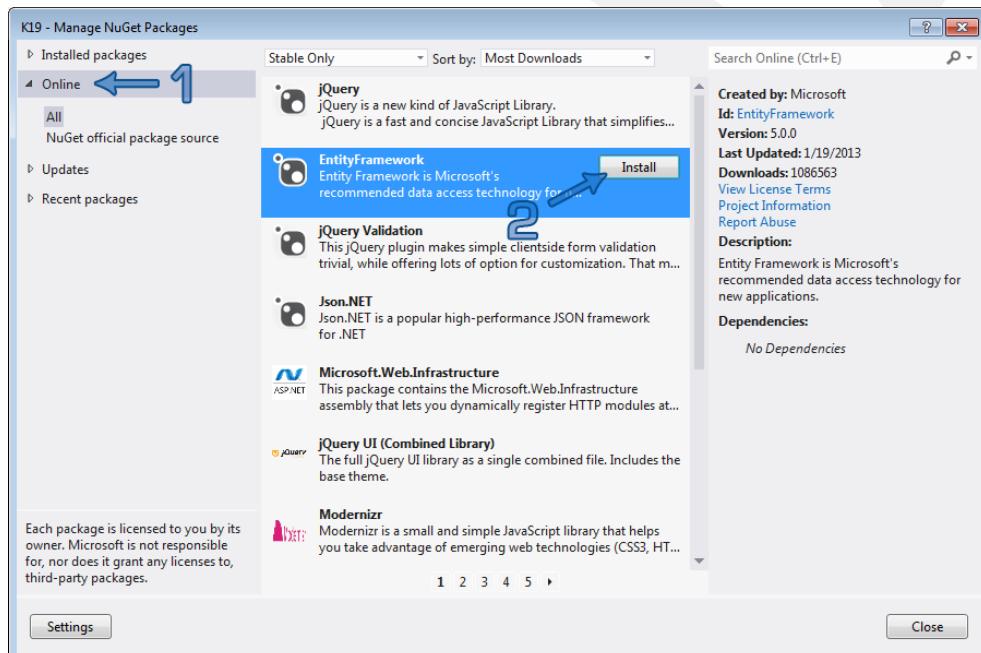


Figura 3.6: Manage NuGet Package

Configuração

O Entity Framework é fortemente baseado no conceito de “Convenção sobre Configuração”. Dessa forma, nenhuma configuração é necessária a não ser que seja necessário alterar o comportamento padrão.

No comportamento padrão do Entity Framework, as conexões com o banco de dados apontarão para `.\SQLEXPRESS`, ou seja, para a instância do SQL Express rodando na mesma máquina que a

aplicação.

Para alterar as configurações de conexão com o banco de dados, podemos defini-las no arquivo de configuração da aplicação. Para aplicações web, devemos alterar o arquivo **Web.config**. Por outro lado, para aplicações desktop, devemos alterar o arquivo **App.config**. Veja o exemplo abaixo.

```

1 <?xml version="1.0" ?>
2 <configuration>
3   <connectionStrings>
4     <add
5       name="K19Context"
6       providerName="System.Data.SqlClient"
7       connectionString="Server=.\SQLEXPRESS;
8         Database=k19db;
9         Trusted_Connection=false;
10        User Id=sa;
11        Password=sa;
12        Persist Security Info=true;
13        MultipleActiveResultSets=True" />
14   </connectionStrings>
15 </configuration>
```

Código XML 3.1: App.config

Quando o Entity Framework é instalado em um projeto do Visual Studio 2012, as configurações padrão são inseridas explicitamente no arquivo de configuração da aplicação. Contudo, essas configurações são opcionais. Inclusive, podemos removê-las e mesmo assim elas serão aplicadas.

Convenções de Mapeamento

Por padrão, no Entity Framework, o mapeamento das entidades segue as regras do **Code First**. Considere as entidades Livro e Editora definidas abaixo.

```

1 public class Livro
2 {
3   public int LivroID { get; set; }
4   public string Titulo { get; set; }
5   public decimal Preco { get; set; }
6   public Editora Editora { get; set; }
7 }
```

Código C# 3.1: Livro.cs

```

1 public class Editora
2 {
3   public int EditoraID { get; set; }
4   public string Nome { get; set; }
5   public string Email { get; set; }
6   public ICollection<Livro> Livros { get; set; }
7 }
```

Código C# 3.2: Editora.cs

As entidades devem ser “registradas” em um **DbContext**. Para registrar uma entidade, basta definir uma propriedade do tipo **DbSet**. Veja o exemplo abaixo.

```

1 public namespace EF
2 {
3   public class K19Context : DbContext
```

```

4     {
5         public DbSet<Editora> Editoras { get; set; }
6         public DbSet<Livro> Livros { get; set; }
7     }
8 }
```

Código C# 3.3: K19Context.cs

De acordo com as convenções do Code First, as entidades **Editora** e **Livro** serão mapeadas para tabelas chamadas **Editoras** e **Livros** respectivamente. Observe que os nomes das entidades foram pluralizados seguindo as regras da língua inglesa para definir os nomes das tabelas.

Além disso, por padrão, essas tabelas serão criadas em uma base de dados chamada **EF.K19Context**. Esse nome corresponde ao *full qualified name* da classe K19Context.



Mais Sobre

Para desabilitar a pluralização aplicada no nome das tabelas, é necessário sobrescrever o método **OnModelCreating()** da classe **DbContext**.

```

1 public class K19Context : DbContext
2 {
3     public DbSet<Editora> Editoras { get; set; }
4     public DbSet<Livro> Livros { get; set; }
5
6     protected override void OnModelCreating( DbModelBuilder dbModelBuilder )
7     {
8         dbModelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
9     }
10 }
```

Código C# 3.4: K19Context.cs



Mais Sobre

Nas próximas versões do Entity Framework, as regras de pluralização poderão ser personalizadas. Dessa forma, será possível, por exemplo, substituir as regras da língua inglesa pelas regras da língua portuguesa.



Mais Sobre

Podemos modificar o nome da base de dados escolhido por padrão. Para isso, basta adicionar um construtor na classe que define o **DbContext** semelhante ao mostrado no código abaixo.

```

1 public class K19Context : DbContext
2 {
3     public DbSet<Editora> Editoras { get; set; }
4     public DbSet<Livro> Livros { get; set; }
5
6     public K19Context() : base("MinhaBaseDeDados")
7     {
8     }
9 }
```

Código C# 3.5: K19Context.cs

Primary Key

O Entity Framework assumirá como identificador de uma entidade, a propriedade cujo nome seja **ID** ou **EntidadeID** escrito com letras maiúsculas e/ou minúsculas. Essa propriedade será mapeada para uma coluna definida no banco de dados como chave primária da tabela correspondente a entidade. Veja os exemplos abaixo.

```
1 public class Livro
2 {
3     public int LivroID { get; set; }
4 }
```

Código C# 3.6: Livro.cs

```
1 public class Livro
2 {
3     public int ID { get; set; }
4 }
```

Código C# 3.7: Livro.cs

Se o tipo da propriedade utilizada como identificador de uma entidade for numérico, os valores da coluna correspondente a essa propriedade serão gerados pelo banco de dados. Isso significa que, no SQL Server, a coluna será definida com a opção **Identity**.

Complex Types

Considere a seguinte entidade.

```
1 public class Contato
2 {
3     public int ContatoID { get; set; }
4
5     public string LogradouroResidencial { get; set; }
6     public string LogradouroComercial { get; set; }
7
8     public string CEPResidencial { get; set; }
9     public string CEPComercial { get; set; }
10
11    public String BairroResidencial { get; set; }
12    public String BairroComercial { get; set; }
13 }
```

Código C# 3.8: Contato.cs

Do ponto de vista da modelagem orientada a objetos, deveríamos refatorar essa classe separando as propriedades relacionadas aos endereços de um contato.

```
1 public class Contato
2 {
3     public int ContatoID { get; set; }
4
5     public Endereco EnderecoResidencial { get; set; }
6     public Endereco EnderecoComercial { get; set; }
7 }
```

Código C# 3.9: Contato.cs

```
1 public class Endereco
2 {
```

```

3     public string Logradouro { get; set; }
4     public string CEP { get; set; }
5     public string Bairro { get; set; }
6 }
```

Código C# 3.10: Endereco.cs

Como nenhuma propriedade da classe Endereco se enquadra nas convenções de *Primary Key* e supondo que nenhuma propriedade tenha sido mapeada explicitamente para uma coluna definida como chave primária, o Entity Framework assumirá que essa classe é um **Complex Type**. Além disso, um Complex Type não pode referenciar uma entidade através de uma propriedade.

As classes Contato e Endereco serão mapeadas para uma única tabela chamada **Contatoes**. Essa tabela possuirá a estrutura mostrada na imagem a seguir.

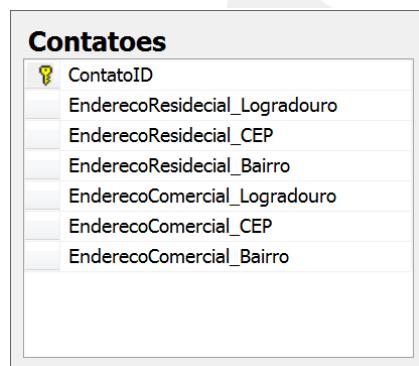


Figura 3.7: Mapeamento das classes Contato e Endereco

Mapeamento dos Relacionamentos

Considere as seguintes entidades.

```

1 public class Livro
2 {
3     public int LivroID { get; set; }
4
5     public Editora Editora { get; set; }
6 }
```

Código C# 3.11: Livro.cs

```

1 public class Editora
2 {
3     public int EditoraID { get; set; }
4 }
```

Código C# 3.12: Editora.cs

Essas duas entidades serão mapeadas para as tabelas Livroes e Editoras. Na tabela Livroes, será adicionada uma chave estrangeira chamada Editora_EditoraID referenciado a chave primária da tabela Editora. Observe na imagem abaixo a estrutura das tabelas.

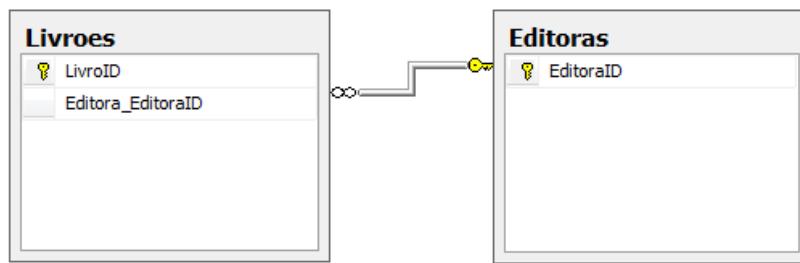


Figura 3.8: Mapeamento das entidades Livro e Editora

As mesmas tabelas seriam obtidas se o relacionamento fosse definido na entidade Editora ao invés de ser definido na entidade Livro.

```

1 public class Livro
2 {
3     public int LivroID { get; set; }
4 }
  
```

Código C# 3.13: Livro.cs

```

1 public class Editora
2 {
3     public int EditoraID { get; set; }
4
5     public ICollection<Livro> Livros { get; set; }
6 }
  
```

Código C# 3.14: Editora.cs

Também obteríamos as mesmas tabelas se o relacionamento fosse definido em ambas entidades.

```

1 public class Livro
2 {
3     public int LivroID { get; set; }
4
5     public Editora Editora { get; set; }
6 }
  
```

Código C# 3.15: Livro.cs

```

1 public class Editora
2 {
3     public int EditoraID { get; set; }
4
5     public ICollection<Livro> Livros { get; set; }
6 }
  
```

Código C# 3.16: Editora.cs

Por outro lado, se na entidade Livro uma coleção de editoras for definida como propriedade e de forma análoga uma coleção de livros for definida como propriedade na entidade Editora então uma terceira tabela será criada no banco de dados para representar esse relacionamento “muitos para muitos” (many to many).

```

1 public class Livro
2 {
  
```

```

3     public int LivroID { get; set; }
4
5     public ICollection<Editora> Editoras { get; set; }
6 }
```

Código C# 3.17: Livro.cs

```

1 public class Editora
2 {
3     public int EditoraID { get; set; }
4
5     public ICollection<Livro> Livros { get; set; }
6 }
```

Código C# 3.18: Editora.cs



Figura 3.9: Mapeamento das entidades Livro e Editora

Type Discovery

Quando duas entidades estão relacionadas, não é necessário que ambas sejam “registradas” no DbContext. Apenas a entidade principal necessita ser registrada. Por exemplo, considere as seguintes entidades.

```

1 public class Livro
2 {
3     public int LivroID { get; set; }
4 }
```

Código C# 3.19: Livro.cs

```

1 public class Editora
2 {
3     public int EditoraID { get; set; }
4
5     public ICollection<Livro> Livros { get; set; }
6 }
```

Código C# 3.20: Editora.cs

No DbContext da aplicação, podemos registrar ambas ou apenas a entidade Editora.

```

1 public class K19Context : DbContext
2 {
3     public DbSet<Editora> Editoras { get; set; }
4     public DbSet<Livro> Livros { get; set; }
5 }
```

Código C# 3.21: K19Context.cs

```

1 public class K19Context : DbContext
2 {
3     public DbSet<Editora> Editoras { get; set; }
```

```
4 }
```

Código C# 3.22: K19Context.cs

Se as duas entidades forem registradas, tanto os livros quanto as editoras podem ser obtidos diretamente pelo DbContext. Agora, se apenas a entidade Editora for registrada, somente as editoras podem ser obtidas diretamente pelo DbContext e os livros seriam obtidos somente através das editoras.



Exercícios de Fixação

- 1 Abra o *Visual Studio Express 2012 for Desktop*; Crie um projeto do tipo **Console Application** em C# chamado **K19EntityFramework**.
- 2 Instale o Entity Framework no projeto **K19EntityFramework** utilizando o **Manage Nuget Package**. Siga o exemplo das imagens abaixo.

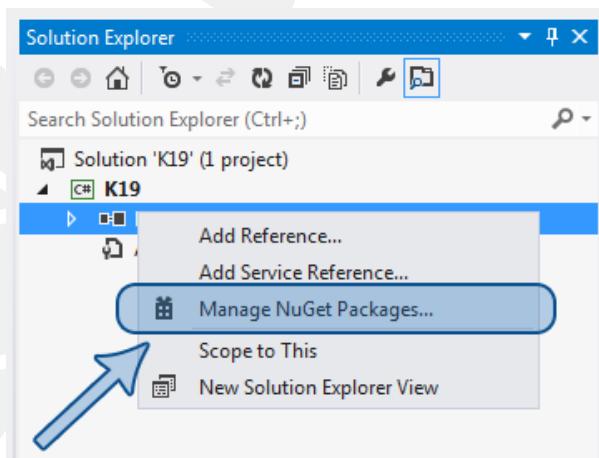


Figura 3.10: Manage NuGet Package

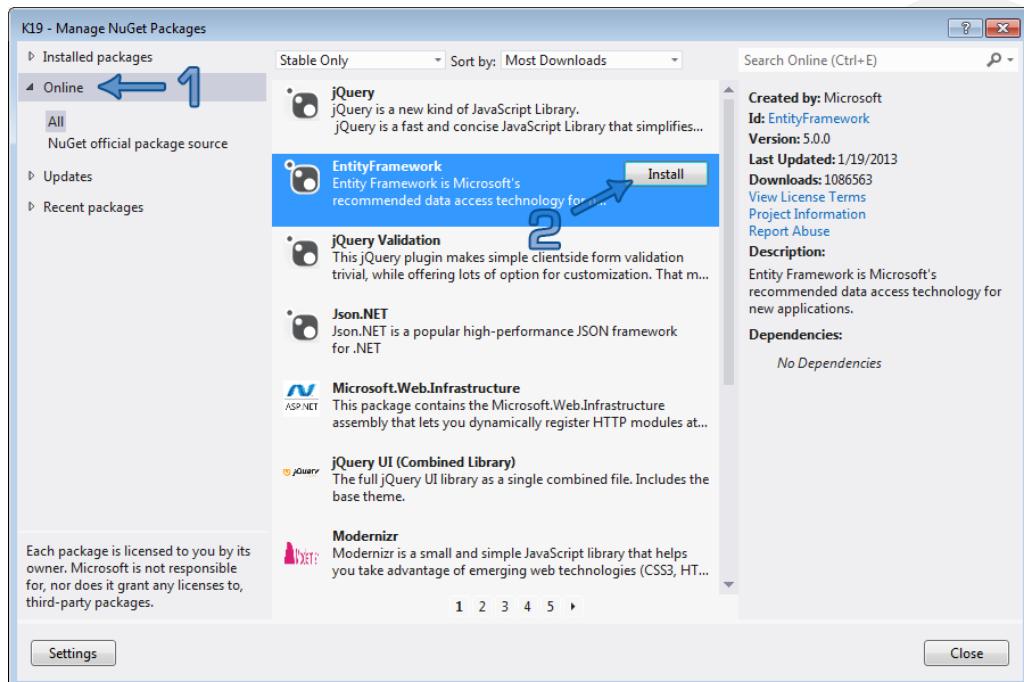


Figura 3.11: Manage NuGet Package

- 3** Adicione uma classe chamada **Endereco** no projeto **K19EntityFramework**.

```

1 namespace K19EntityFramework
2 {
3     // complex type
4     public class Endereco
5     {
6         public string Logradouro { get; set; }
7
8         public int Numero { get; set; }
9
10        public string CEP { get; set; }
11    }
12 }
```

Código C# 3.23: Endereco.cs

- 4** Adicione uma classe chamada **Aluno** no projeto **K19EntityFramework**.

```

1 namespace K19EntityFramework
2 {
3     public class Aluno
4     {
5         public int AlunoID { get; set; }
6
7         public string Nome { get; set; }
8
9         public Endereco Endereco { get; set; }
10    }
11 }
```

Código C# 3.24: Aluno.cs

- 5 Adicione uma classe chamada **Professor** no projeto **K19EntityFramework**.

```

1 namespace K19EntityFramework
2 {
3     public class Professor
4     {
5         public int ProfessorID { get; set; }
6
7         public string Nome { get; set; }
8
9         public Endereco Endereco { get; set; }
10    }
11 }
```

Código C# 3.25: Professor.cs

- 6 Adicione uma classe chamada **Turma** no projeto **K19EntityFramework**.

```

1 namespace K19EntityFramework
2 {
3     public class Turma
4     {
5         public int TurmaId { get; set; }
6
7         public int Vagas { get; set; }
8
9         public Professor Professor { get; set; }
10
11        public ICollection<Aluno> Alunos { get; set; }
12    }
13 }
```

Código C# 3.26: Turma.cs

- 7 Adicione uma classe chamada **K19Context** no projeto **K19EntityFramework**.

```

1 namespace K19EntityFramework
2 {
3     public class K19Context : DbContext
4     {
5         public DbSet<Turma> Turmas { get; set; }
6
7         public DbSet<Aluno> Alunos { get; set; }
8
9         public DbSet<Professor> Professores { get; set; }
10    }
11 }
```

Código C# 3.27: Turma.cs

- 8 Adicione uma classe chamada **Teste** no projeto **K19EntityFramework**.

```

1 namespace K19EntityFramework
2 {
3     public class Teste
4     {
5         static void Main(string[] args)
6         {
7             using (var ctx = new K19Context())
8             {
9                 Aluno a1 = new Aluno
```

```
10      {
11          Nome = "Amanda Guerra",
12          Endereco = new Endereco
13          {
14              Logradouro = "Rua Amalera",
15              Numero = 1789,
16              CEP = "00157-001"
17          }
18      };
19
20      Aluno a2 = new Aluno
21      {
22          Nome = "Marcelo Martins",
23          Endereco = new Endereco
24          {
25              Logradouro = "Rua Zaul",
26              Numero = 2907,
27              CEP = "09147-001"
28          }
29      };
30
31      Professor p = new Professor
32      {
33          Nome = "Jonas Hirata",
34          Endereco = new Endereco
35          {
36              Logradouro = "Rua Movelha",
37              Numero = 8367,
38              CEP = "00876-100"
39          }
40      };
41
42      Turma t = new Turma
43      {
44          Vagas = 10,
45          Alunos = new List<Aluno>()
46      };
47
48      t.Professor = p;
49      t.Alunos.Add(a1);
50      t.Alunos.Add(a2);
51
52      ctx.Turmas.Add(t);
53      ctx.SaveChanges();
54  }
55 }
56 }
```

Código C# 3.28: Teste.cs

- 9 Execute o projeto **K19EntityFramework** e consulte o banco de dados para verificar se os dados foram armazenados corretamente. Observe a estrutura das tabelas.

Data Annotations

Podemos utilizar anotações para sobrescrever as convenções de mapeamento. A seguir apresentaremos as principais anotações.

Table e Column

No Entity Framework, os nomes das tabelas e das colunas são definidos por convenção. Contudo, podemos personalizá-los de acordo com a necessidade através das anotações **Table** e **Column**.

```

1 [Table("tbl_alunos")]
2 public class Aluno
3 {
4     public int AlunoID { get; set; }
5
6     [Column("nome_do_aluno")]
7     public string Nome { get; set; }
8 }
```

Código C# 3.29: Aluno.cs

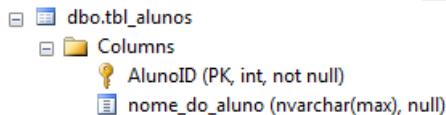


Figura 3.12: Personalizando os nomes das tabelas e das colunas

A anotação **Column** também permite a escolha do tipo da coluna correspondente à propriedade anotada.

```

1 public class Produto
2 {
3     public int ProdutoID { get; set; }
4
5     [Column("descricao_do_produto", TypeName="text")]
6     public string Descricao { get; set; }
7 }
```

Código C# 3.30: Produto.cs

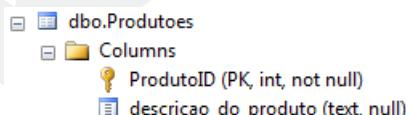


Figura 3.13: Personalizando o tipo de uma coluna

DatabaseGenerated

Por padrão, quando a propriedade correspondente à chave primária de uma tabela é numérica, os valores dessa chave serão gerados pelo banco de dados. Podemos alterar esse comportamento através da anotação **DatabaseGenerated**.

```

1 public class Aluno
2 {
3     [DatabaseGenerated(DatabaseGeneratedOption.None)]
4     public int AlunoID { get; set; }
5
6     public string Nome { get; set; }
7 }
```

Código C# 3.31: Aluno.cs

Key

Considere a seguinte classe.

```

1 public class Aluno
2 {
3     public int Código { get; set; }
4
5     public string Nome { get; set; }
6 }
```

Código C# 3.32: Aluno.cs

Nenhuma propriedade se enquadra nas convenções de nomenclatura de chave primária. Contudo, é possível definir explicitamente, qual propriedade será mapeada como chave primária através da anotação **Key**. No exemplo abaixo, a propriedade Código foi definida como chave da entidade.

```

1 public class Aluno
2 {
3     [Key]
4     public int Código { get; set; }
5
6     public string Nome { get; set; }
7 }
```

Código C# 3.33: Aluno.cs



Figura 3.14: Escolhendo a chave primária explicitamente

Required

Uma propriedade pode ser definida explicitamente como obrigatória através da anotação **Required**. No banco de dados, as colunas correspondentes às propriedades obrigatórias serão definidas com a opção **not null**.

```

1 public class Aluno
2 {
3     public int AlunoID { get; set; }
4
5     [Required]
6     public string Nome { get; set; }
7 }
```

Código C# 3.34: Aluno.cs

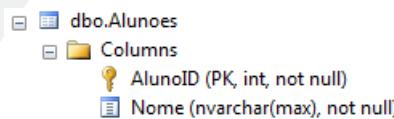


Figura 3.15: Definindo propriedades obrigatórias

MaxLength e MinLength

```

1 public class Aluno
2 {
3     public int AlunoID { get; set; }
4
5     [MaxLength(30), MinLength(3)]
6     public string Nome { get; set; }
7 }
```

Código C# 3.35: Aluno.cs

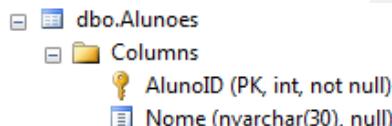


Figura 3.16: Definindo a quantidade mínima e máxima de caracteres de uma propriedade

NotMapped

Considere a seguinte classe.

```

1 public class Aluno
2 {
3     public int AlunoID { get; set; }
4
5     public DateTime DataDeNascimento { get; set; }
6
7     public int Idade { get; set; }
8
9 }
```

Código C# 3.36: Aluno.cs

Observe que a propriedade `Idade` pode ser calculada a partir da propriedade `DataDeNascimento`. Portanto, não é necessário armazenar o valor da idade no banco de dados. Nesses casos, podemos utilizar a anotação `NotMapped` e indicar quais propriedades não devem ser mapeadas nas tabelas.

```

1 public class Aluno
2 {
3     public int AlunoID { get; set; }
4
5     public DateTime DataDeNascimento { get; set; }
6
7     [NotMapped]
8     public int Idade { get; set; }
9
10 }
```

Código C# 3.37: Aluno.cs

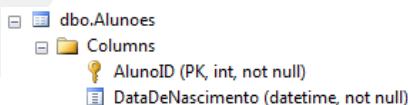


Figura 3.17: Excluindo uma propriedade no mapeamento

InverseProperty

Considere as seguintes entidades.

```

1 public class Pessoa
2 {
3     public int PessoaID { get; set; }
4
5     public string Nome { get; set; }
6
7     public ICollection<Livro> LivrosPublicados { get; set; }
8
9     public ICollection<Livro> LivrosRevisados { get; set; }
10}

```

Código C# 3.38: Pessoa.cs

```

1 public class Livro
2 {
3     public int LivroID { get; set; }
4
5     public string Titulo { get; set; }
6
7     public Pessoa Autor { get; set; }
8
9     public Pessoa Revisor { get; set; }
10}

```

Código C# 3.39: Livro.cs

Nesse caso, como o Entity Framework não consegue combinar as propriedades das duas entidades para estabelecer dois relacionamentos bidirecionais, ele criará no banco de dados quatro chaves estrangeiras, uma para cada propriedade de navegação das entidades Pessoa e Livro.

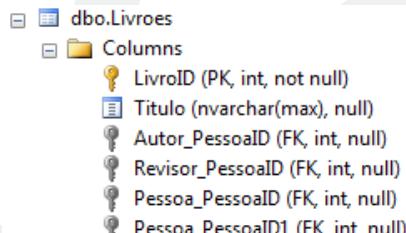


Figura 3.18: Relacionamentos não combinados

Para agrupar as propriedades de navegação das entidades Pessoa e Livro, devemos utilizar a anotação **InverseProperty**. Observe o código a seguir.

```

1 public class Pessoa
2 {
3     public int PessoaID { get; set; }
4
5     public string Nome { get; set; }
6
7     [InverseProperty("Autor")]
8     public ICollection<Livro> LivrosPublicados { get; set; }
9
10    [InverseProperty("Revisor")]
11    public ICollection<Livro> LivrosRevisados { get; set; }
12}

```

Código C# 3.40: Pessoa.cs

```
1 public class Livro
```

```

2  {
3      public int LivroID { get; set; }
4
5      public string Titulo { get; set; }
6
7      public Pessoa Autor { get; set; }
8
9      public Pessoa Revisor { get; set; }
10 }

```

Código C# 3.41: Livro.cs

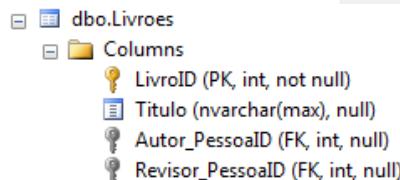


Figura 3.19: Relacionamentos combinados com InverseProperty



Exercícios de Fixação

- 10 Altere a classe **Aluno** do projeto **K19EntityFramework**.

```

1  namespace K19EntityFramework
2  {
3      [Table("tbl_alunos")]
4      public class Aluno
5      {
6          public int AlunoID { get; set; }
7
8          [Required]
9          public string Nome { get; set; }
10
11         public Endereco Endereco { get; set; }
12     }
13 }

```

Código C# 3.42: Aluno.cs

- 11 Apague a base de dados **K19EntityFramework.K19Context** e execute novamente o projeto. Observe as modificações na estrutura da tabela **Alunoes** que passará a se chamar **tbl_alunos**. Observe que agora a coluna **Nome** não pode possuir valores nulos.

Gerenciamento de entidades

As instâncias das entidades de uma aplicação que utiliza o Entity Framework são administradas por objetos do tipo `DbContext`. Esses objetos são responsáveis pelas operações de inserção, remoção, alteração e consulta. A seguir, mostraremos o funcionamento básico de um `DbContext`.

A primeira regra básica sobre os DbContexts é sempre utilizá-los dentro de um bloco `using` para evitar que os recursos utilizados por eles não sejam devidamente fechados.

```
1 using (var context = new K19Context())
2 {
3     // utilizando o DbContext
4 }
```

Código C# 3.43: Utilizando um DbContext dentro do bloco using

Persistindo

Para indicar que determinado objeto deve ser persistido no banco de dados, devemos utilizar o método `System.Data.Entity.DbSet.Add()`, passando o objeto em questão como parâmetro. O método `Add()` adiciona o objeto ao contexto com o estado **Added**.

Para armazenar de fato as informações de um objeto no banco de dados, utilizamos o método `SaveChanges()` do `DbContext`. Todos os objetos do contexto que estejam no estado `Added` são inseridos no banco de dados quando o método `SaveChanges()` é chamado.

```
1 Editora editora = new Editora { Nome = "K19", Email = "contato@k19.com.br" };
2
3 // Adiciona a editora criada ao contexto com estado Added
4 context.Editoras.Add(editora);
5
6 // Sincroniza o contexto com o banco de dados
7 context.SaveChanges();
```

Código C# 3.44: Persistindo um objeto

Buscando

Para obter um objeto que contenha informações do banco de dados, basta utilizar o método `Find()` do `DbSet`. Devemos passar o identificador do objeto que desejamos recuperar como parâmetro para esse método.

```
1 Editora editora = context.Editoras.Find(1);
```

Código C# 3.45: Buscando uma editora com o identificador 1

Removendo

Para indicar que determinado objeto deve ser removido, basta utilizar o método `System.Data.Entity.DbSet.Remove()`. Esse método marca o objeto para ser removido do banco de dados, colocando-o no estado **Deleted**.

Quando o método `SaveChanges()` é chamado, todas os objetos no estado `Deleted` são removidos do banco de dados.

```
1 Editora editora = context.Editoras.Find(1);
2
3 // Marca a editora que deve ser removida
4 context.Editoras.Remove(editora);
5
6 // Sincroniza o contexto com o banco de dados
7 context.SaveChanges();
```

Código C# 3.46: Removendo uma editora do banco de dados

Atualizando

Para alterar os dados do registro correspondente a um objeto, basta modificar as suas propriedades. Quando as propriedades de um objeto do contexto são alteradas, o estado **Modified** é associado a este objeto. Objetos no estado Modified são atualizados no banco de dados quando o método `SaveChanges()` é chamado.

```

1 Editora editora = context.Editors.Find(1);
2
3 // Modificando as propriedades de um objeto
4 editora.Nome = "Abril S/A";
5
6 // Sincroniza o contexto com o banco de dados
7 context.SaveChanges();

```

Código C# 3.47: Alterando o nome de uma editora

Listando

Para obter uma listagem com todos os objetos referentes aos registros de uma tabela, podemos utilizar a Language Integrated Query (LINQ), que nos permite escrever consultas dentro do código C#.

```

1 var consulta = from e in context.Editors
2                 where e.Nome.Contains("A")
3                 select e;
4
5 // Equivalente a: SELECT * FROM Editors e where e.Nome Like 'A%'
6 foreach (var item in consulta)
7 {
8     System.Console.WriteLine("Editora: " + item.Nome);
9 }

```

Código C# 3.48: Utilizando LINQ para fazer uma consulta



Exercícios de Fixação

- 12 Adicione uma classe no projeto **K19EntityFramework** chamada **InsereAluno**.

```

1 namespace K19EntityFramework
2 {
3     public class InsereAluno
4     {
5         static void Main(string[] args)
6         {
7             using (var ctx = new K19Context())
8             {
9                 Console.WriteLine("Digite o nome do aluno");
10                string nome = Console.ReadLine();
11
12                Console.WriteLine("Digite o logradouro do endereço do aluno");
13                string logradouro = Console.ReadLine();
14

```

```

15     Console.WriteLine("Digite o número do endereço do aluno");
16     int numero = Int32.Parse(Console.ReadLine());
17
18     Console.WriteLine("Digite o CEP do endereço do aluno");
19     string cep = Console.ReadLine();
20
21     Aluno a = new Aluno
22     {
23         Nome = nome,
24         Endereco = new Endereco
25         {
26             Logradouro = logradouro,
27             Numero = numero,
28             CEP = cep
29         }
30     };
31
32     ctx.Alunos.Add(a);
33     ctx.SaveChanges();
34
35 }
36
37 }
38 }
```

Código C# 3.49: InsereAluno.cs

Nas propriedades do projeto K19EntityFramework, altere o startup object selecionando a classe **InsereAluno. Depois execute o projeto.**

- 13** Adicione uma classe no projeto **K19EntityFramework** chamada **ListaAluno**.

```

1  using System.Linq;
2  namespace K19EntityFramework
3  {
4      public class ListaAluno
5      {
6          static void Main(string[] args)
7          {
8              using (var ctx = new K19Context())
9              {
10                  var consulta = from a in ctx.Alunos select a;
11
12                  foreach (Aluno a in consulta)
13                  {
14                      Console.WriteLine("Nome: " + a.Nome);
15                      Console.WriteLine("Logradouro: " + a.Endereco.Logradouro);
16                      Console.WriteLine("Número: " + a.Endereco.Numero);
17                      Console.WriteLine("CEP: " + a.Endereco.CEP);
18                      Console.WriteLine("-----");
19                  }
20              }
21          }
22      }
23 }
```

Código C# 3.50: ListaAluno.cs

Nas propriedades do projeto K19EntityFramework, altere o startup object selecionando a classe **ListaAluno. Depois execute o projeto.**

- 14** Adicione uma classe no projeto **K19EntityFramework** chamada **AlteraAluno**.

```

1  namespace K19EntityFramework
2  {
3      public class AlteraAluno
4      {
5          static void Main(string[] args)
6          {
7              using (var ctx = new K19Context())
8              {
9                  Aluno a = ctx.Alunos.Find(1);
10                 a.Nome = a.Nome + "ALTERADO";
11
12                 ctx.SaveChanges();
13             }
14         }
15     }
16 }
```

Código C# 3.51: AlteraAluno.cs

Nas propriedades do projeto K19EntityFramework, altere o startup object selecionando a classe AlteraAluno. Depois execute o projeto.

- 15 Adicione uma classe no projeto **K19EntityFramework** chamada **RemoveAluno**.

```

1  namespace K19EntityFramework
2  {
3      public class RemoveAluno
4      {
5          static void Main(string[] args)
6          {
7              using (var ctx = new K19Context())
8              {
9                  Aluno a = ctx.Alunos.Find(1);
10
11                  ctx.Alunos.Remove(a);
12
13                  ctx.SaveChanges();
14             }
15         }
16     }
17 }
```

Código C# 3.52: RemoveAluno.cs

Nas propriedades do projeto K19EntityFramework, altere o startup object selecionando a classe RemoveAluno. Depois execute o projeto.

Repositórios

As classes `DbContext` e `DbSet` do Entity Framework oferecem recursos suficientes para que os objetos do domínio sejam persistidos, recuperados, alterados e removidos do banco de dados. Porém, em aplicações com alta complexidade e grande quantidade de código, “espalhar” as chamadas aos métodos do `DbContext` e `DbSet` pode gerar dificuldades na manutenção e no entendimento do sistema.

Para melhorar a organização das nossas aplicações, diminuindo o custo de manutenção e aumentando a legibilidade do código, podemos aplicar o padrão **Repository** do **DDD** (Domain Driven

Design).

Conceitualmente, um repositório representa o conjunto de todos os objetos de um determinado tipo. Ele deve oferecer métodos para administrar seus elementos.

Os repositórios podem trabalhar com objetos prontos na memória ou reconstruí-los com dados obtidos de um banco de dados. O acesso ao banco de dados pode ser realizado através de uma ferramenta ORM como o Entity Framework.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Data.Entity;
5
6 namespace K19
7 {
8     public class EditoraRepository
9     {
10         DbContext context;
11
12         public EditoraRepository(DbContext context)
13         {
14             this.context = context;
15         }
16
17         public void Adiciona(Editora e)
18         {
19             context.Set<Editora>().Add(e);
20             context.SaveChanges();
21         }
22
23         public Editora Busca(int id)
24         {
25             return context.Set<Editora>().Find(id);
26         }
27
28         public List<Editora> BuscaTodas()
29         {
30             var consulta = from e in context.Set<Editora>()
31                         select e;
32             return consulta.ToList();
33         }
34     }
35 }
```

Código C# 3.53: EditoraRepository.cs

```
1 var context = new K19Context();
2 EditoraRepository repository = new EditoraRepository(context);
3 List<Editora> editoras = repository.BuscaTodas();
```

Código C# 3.54: Buscando todas as editoras armazenadas no banco de dados

VISÃO GERAL DO ASP.NET MVC

Necessidades de uma aplicação web

HTTP

Os usuários de uma aplicação web utilizam navegadores (*browsers*) para interagir com essa aplicação. A comunicação entre navegadores e uma aplicação web é realizada através de requisições e respostas definidas pelo protocolo HTTP. Dessa forma, os desenvolvedores de aplicações web devem estar preparados para trabalhar com o protocolo HTTP.

Acesso simultâneo

Além disso, na grande maioria dos casos, as aplicações web devem ser acessadas por diversos usuários ao mesmo tempo. Consequentemente, os desenvolvedores web devem criar ou utilizar algum mecanismo eficiente que permita esse tipo de acesso.

Conteúdo dinâmico

As páginas de uma aplicação web devem ser geradas dinamicamente. Por exemplo, quando um usuário de uma aplicação de email acessa a sua caixa de entrada, ele deseja ver todos os emails enviados até aquele momento. A página contendo a lista de emails deve ser gerada novamente toda vez que essa página for requisitada. Consequentemente, os desenvolvedores web devem criar ou utilizar algum mecanismo eficiente que permita que o conteúdo das páginas das aplicações web seja gerado dinamicamente.

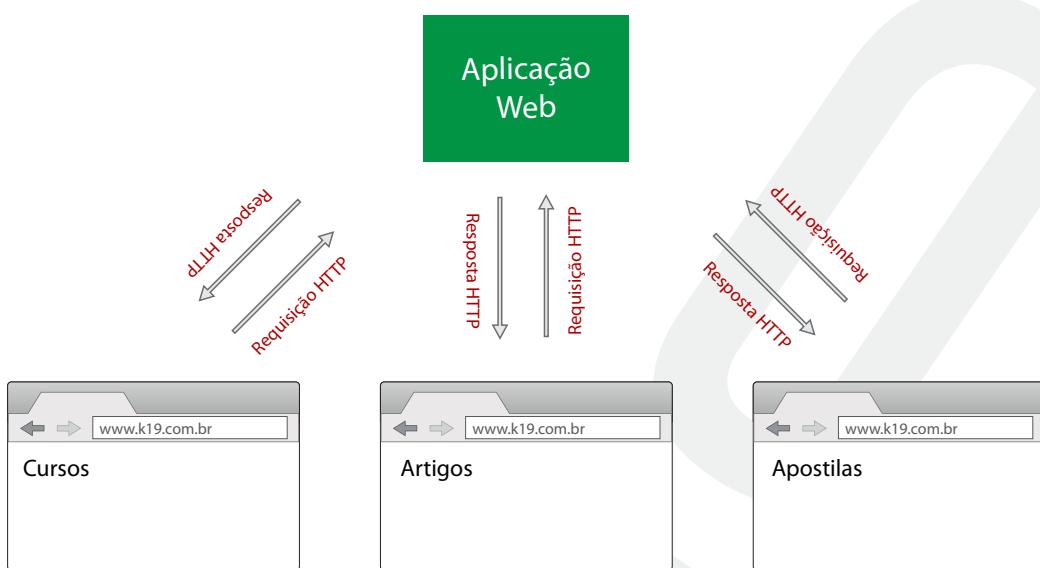


Figura 4.1: Necessidades de uma aplicação web

Solução

Resolver os três problemas apresentados tomaria boa parte do tempo de desenvolvimento, além de exigir conhecimentos técnicos extremamente específicos por parte dos desenvolvedores. Para facilitar o desenvolvimento de aplicações web, a plataforma .NET soluciona esses problemas fundamentais.

A plataforma .NET oferece os seguintes recursos fundamentais para o desenvolvimento de aplicações web:

- Envio e recebimento de mensagens HTTP.
- Acesso simultâneo das aplicações por vários usuários de uma maneira eficiente.
- Geração dinâmica das páginas das aplicações.

ASP.NET MVC

Atualmente, o ASP.NET MVC é o framework para desenvolvimento de aplicações web na plataforma .NET em maior ascensão. A documentação desse framework pode ser obtida em <http://www.asp.net/mvc>. O ASP.NET MVC é fortemente baseado nos padrões *MVC* e *Front Controller*.

MVC e Front Controller

O MVC (*model-view-controller*) é um padrão de arquitetura que tem por objetivo isolar a lógica de negócios da lógica de apresentação de uma aplicação.

Esse padrão (ou alguma variação) é amplamente adotado nas principais plataformas de desenvolvimento atuais. Em particular, ele é bastante utilizado no desenvolvimento de aplicações web.

O padrão MVC divide uma aplicação em três tipos de componentes: modelo, visão e controlador.

Modelo: encapsula os dados e as funcionalidades da aplicação.

Visão: é responsável pela exibição de informações, cujos dados são obtidos do modelo.

Controlador: recebe as requisições do usuário e aciona o modelo e/ou a visão.

Para mais detalhes sobre o padrão MVC, uma boa referência é o livro *Pattern-Oriented Software Architecture Volume 1: A System of Patterns* (editora Wiley, 1996) dos autores Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal e Michael Stal.

No padrão *Front Controller*, todas as requisições do usuário são recebidas pelo mesmo componente. Dessa forma, tarefas que devem ser realizadas em todas as requisições podem ser implementadas por esse componente. Isso evita a repetição de código e facilita a manutenção do sistema.

Para mais informações sobre esse padrão, consulte, por exemplo, o livro *Professional ASP.NET Design Patterns* (editora Wrox, 720) de Scott Millett.

Visual Studio

O primeiro passo para construir uma aplicação web utilizando o framework *ASP.NET MVC 4* é criar um projeto no *Visual Studio* a partir do modelo adequado. No nosso caso, o modelo de projeto que deve ser utilizado é o **ASP.NET MVC 4 Web Application**.

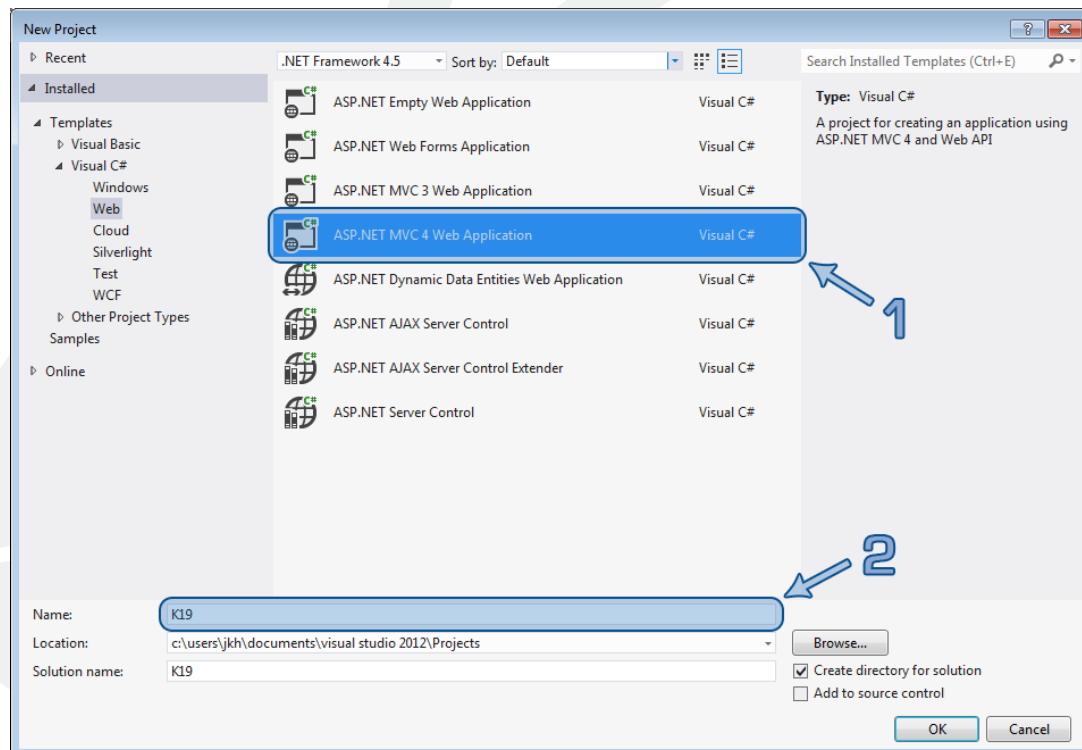


Figura 4.2: Criando um projeto

Devemos escolher *Basic Project* conforme figura abaixo:

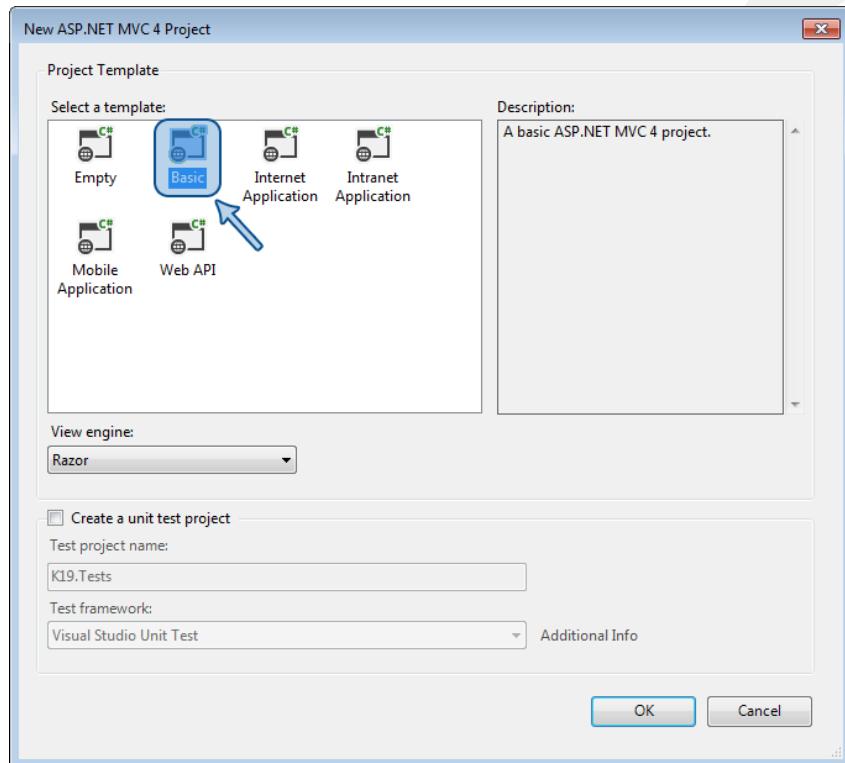


Figura 4.3: Criando um projeto

O projeto criado já vem com diversas pastas e arquivos. Ao longo dos próximos capítulos, a função de cada pasta e de cada arquivo será discutida.

Testando a aplicação

Para verificar o funcionamento do projeto, basta executá-lo através do menu: *Debug -> Start Debugging*. Um servidor será inicializado na máquina e a aplicação é implantada nesse servidor. Além disso, uma janela do navegador padrão do sistema é aberta na url principal da aplicação.

Trocando a porta do servidor

Para trocar a porta do servidor inicializado pelo *Visual Studio* utiliza, basta alterar as propriedades do projeto clicando com o botão direito do mouse no projeto e escolhendo o item *properties* e depois a aba *web*.

Exemplo de uma Aplicação Web

Veremos a seguir um exemplo de uma simples aplicação web em ASP.NET MVC 4. Essa aplicação possuirá uma página principal que deverá exibir o seu número de visualização.

Primeiramente, criaremos um controlador para gerenciar o número de visualização da página principal. Os controladores são definidos por classes que derivam da classe *Controller*. Os nomes

dos controladores devem terminar com a palavra **Controller**. Além disso, eles devem ser definidos na pasta **Controllers** da aplicação. Considere a classe **K19Controller** definida abaixo.

```

1 public class K19Controller : Controller
2 {
3     private static int NumeroDeAcessos { get; set; }
4
5     public ActionResult Index()
6     {
7         K19Controller.NumeroDeAcessos++;
8         ViewBag.Acessos = K19Controller.NumeroDeAcessos;
9         return View();
10    }
11 }
```

Código C# 4.1: K19Controller.cs

A classe **K19Controller** possui uma propriedade chamada **NumeroDeAcessos**. Essa propriedade armazenará o número de visualizações da página principal.

O método **Index()** será chamado toda vez que a url `http://<IP_SERVIDOR>:<PORTA_APP>/K19/Index` for requisitada por um navegador. Note que essa url é formada pelo nome do controlador (nome da classe que define o controlador sem o sufixo **Controller**) seguido pelo nome do método.

Cada vez que o método **Index()** é chamado, o valor da propriedade **NumeroDeAcessos** é incrementado. Além disso, o valor atualizado dessa propriedade é colocado na **ViewBag** para que possa ser acessado na camada de apresentação.

Para indicar que o fluxo de execução deve seguir para a camada de apresentação, o método **Index()** invoca o método **View()** e devolve a resposta obtida. Por padrão, o fluxo será direcionado para um arquivo chamado **Index.cshtml** que deve estar localizado na pasta **Views\K19**.

Na camada de apresentação, vamos adicionar a página associada ao método **Index()**. Para isso, criaremos um arquivo chamado **Index.cshtml** na pasta **Views\K19** da aplicação com o conteúdo abaixo.

```

1 @{
2     ViewBag.Title = "Index";
3 }
4 <h2>Acessos: @ViewBag.Acessos</h2>
```

Código CSHTML 4.1: Index.cshtml

Para acessar a **ViewBag**, devemos utilizar o caractere **@**, como no código acima.



Exercícios de Fixação

- 1 Crie um projeto do tipo **ASP.NET MVC 4 Web Application** chamado **K19** no **Visual Studio 2012 Express for Web**. Utilize o template **Basic Project**.

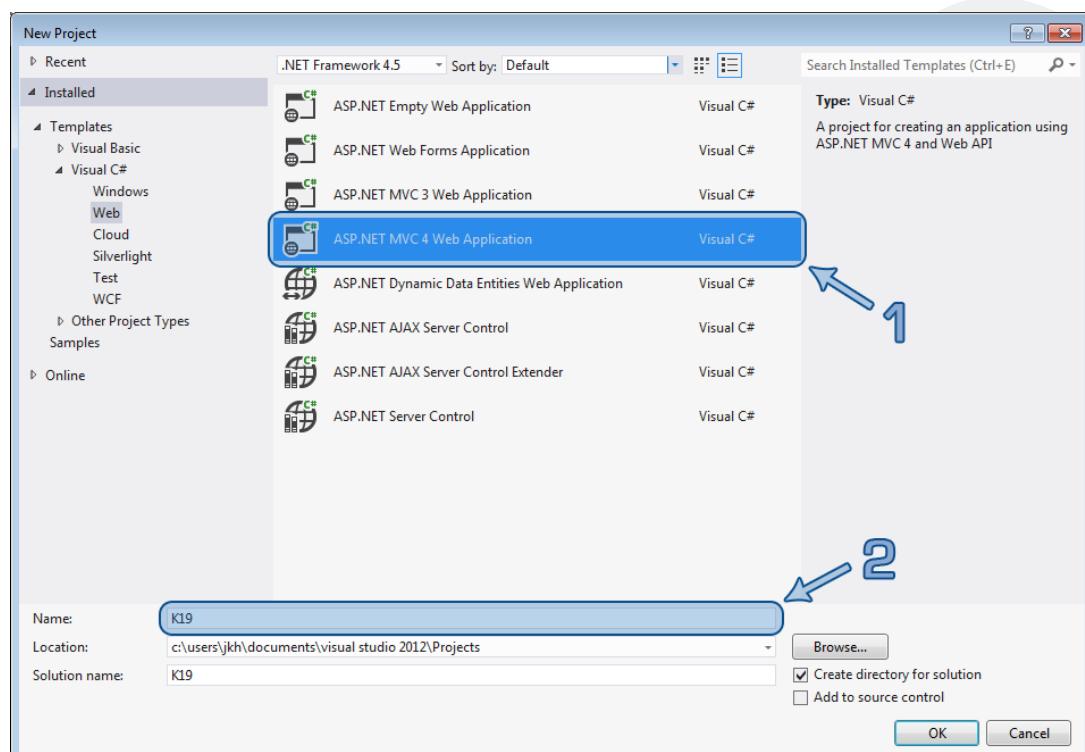


Figura 4.4: Criando um projeto

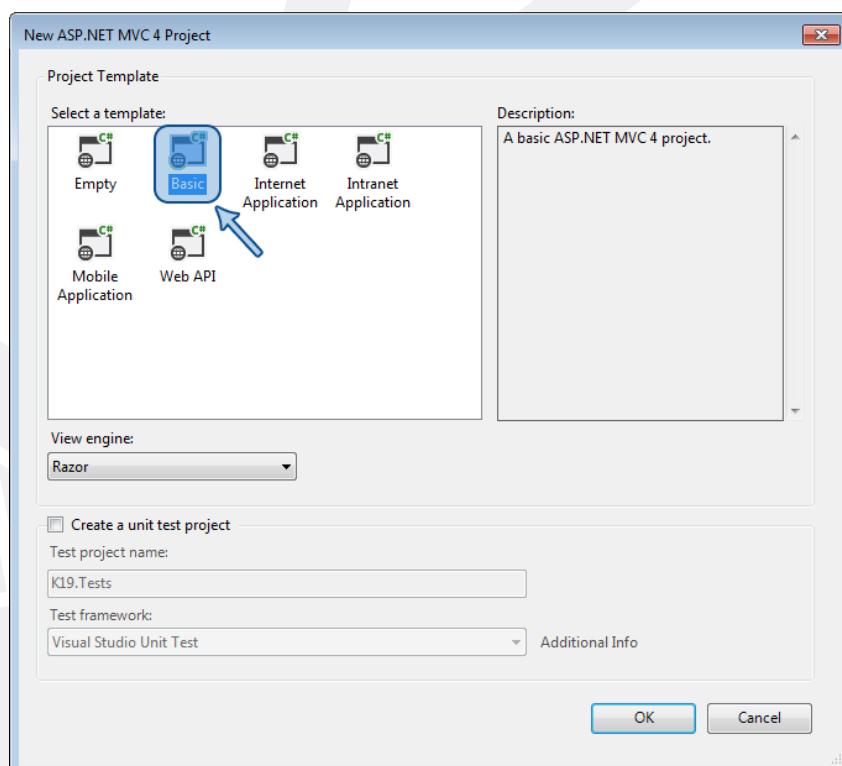


Figura 4.5: Criando um projeto

- 2 Crie um controlador chamado K19 no projeto **K19**. Siga os passos apresentados nas imagens abaixo.

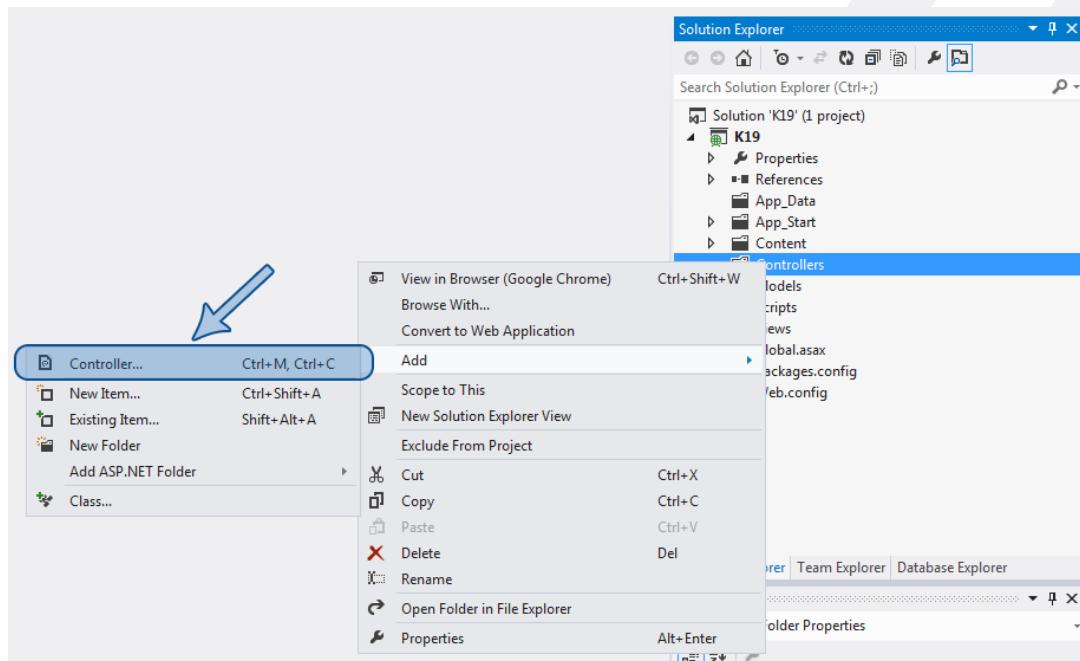


Figura 4.6: Criando um controlador

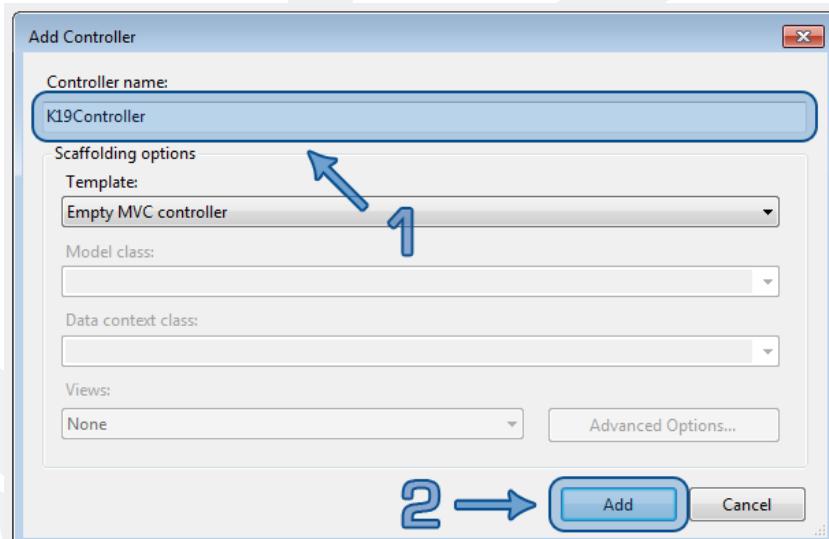


Figura 4.7: Criando um controlador

```

1 using System;
2 using System.Web.Mvc;
3
4 namespace K19.Controllers
5 {
6     public class K19Controller : Controller
7     {
8         public ActionResult Index()
9         {

```

```
10     Random random = new Random();
11     ViewBag.NumeroDaSorte = random.Next();
12 }
13 }
14 }
15 }
```

Código C# 4.2: K19Controller.cs

- 3 Dentro da pasta Views, crie uma pasta chamada K19.

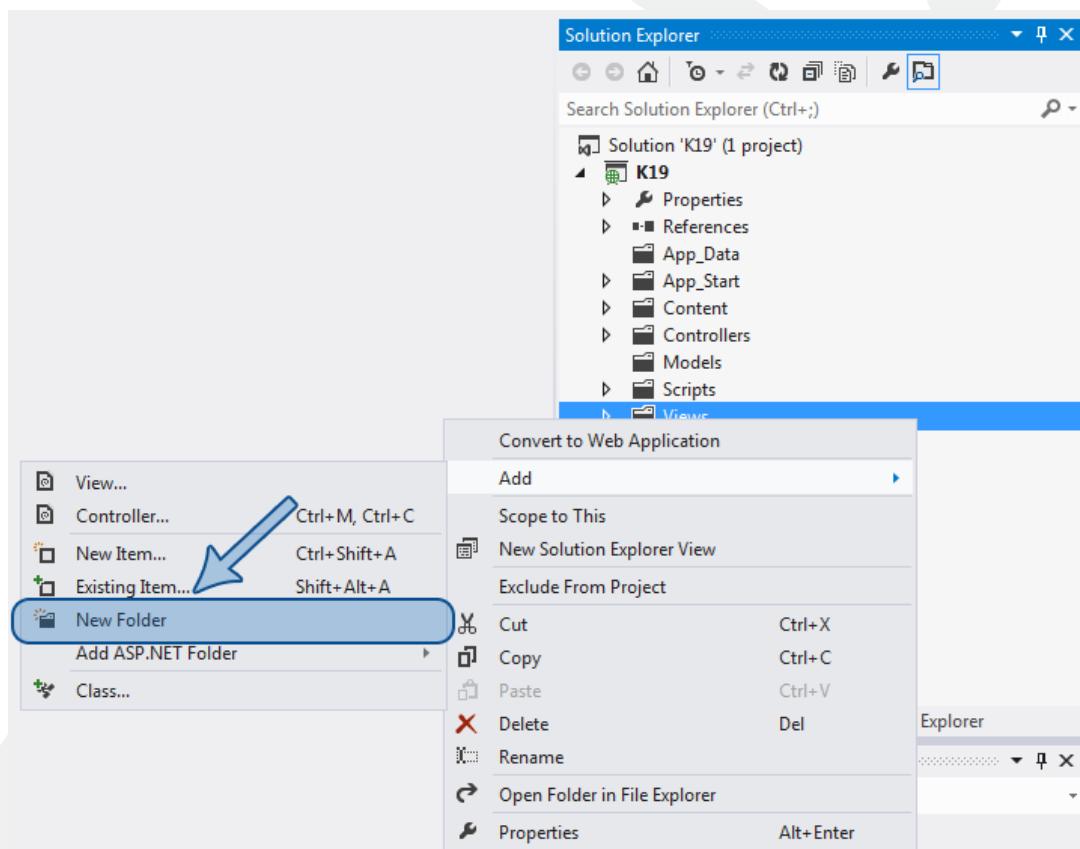


Figura 4.8: Criando uma pasta

- 4 Crie uma página na aplicação, adicionando um arquivo chamado **Index.cshtml** dentro da pasta **Views\K19**.

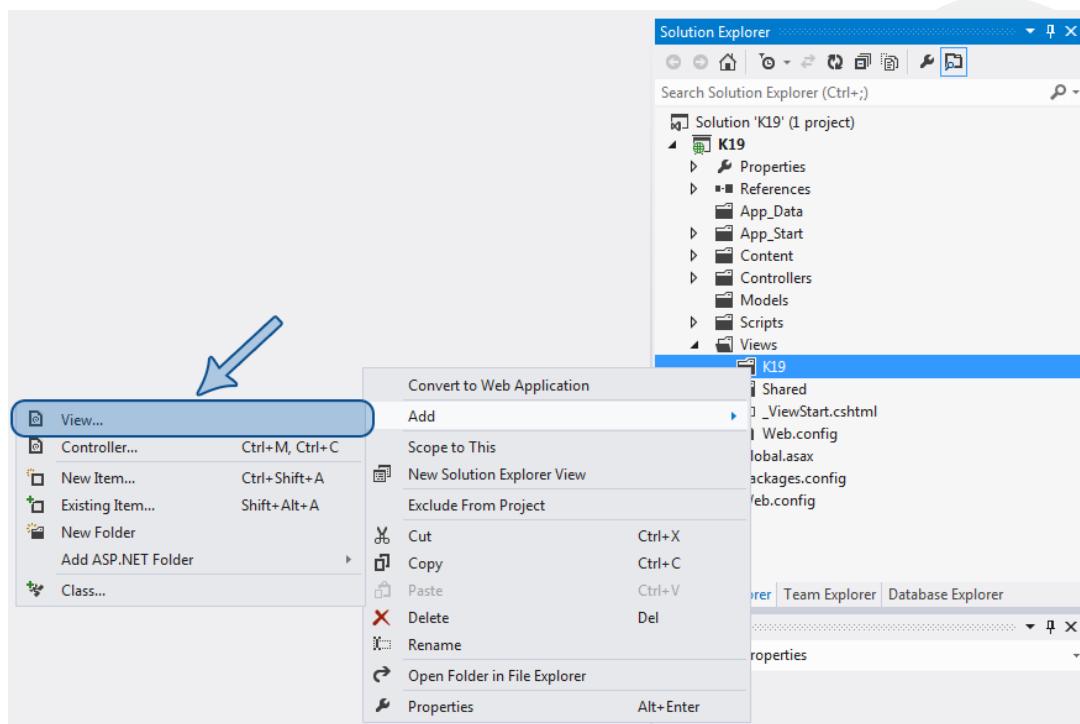


Figura 4.9: Adicionando uma tela

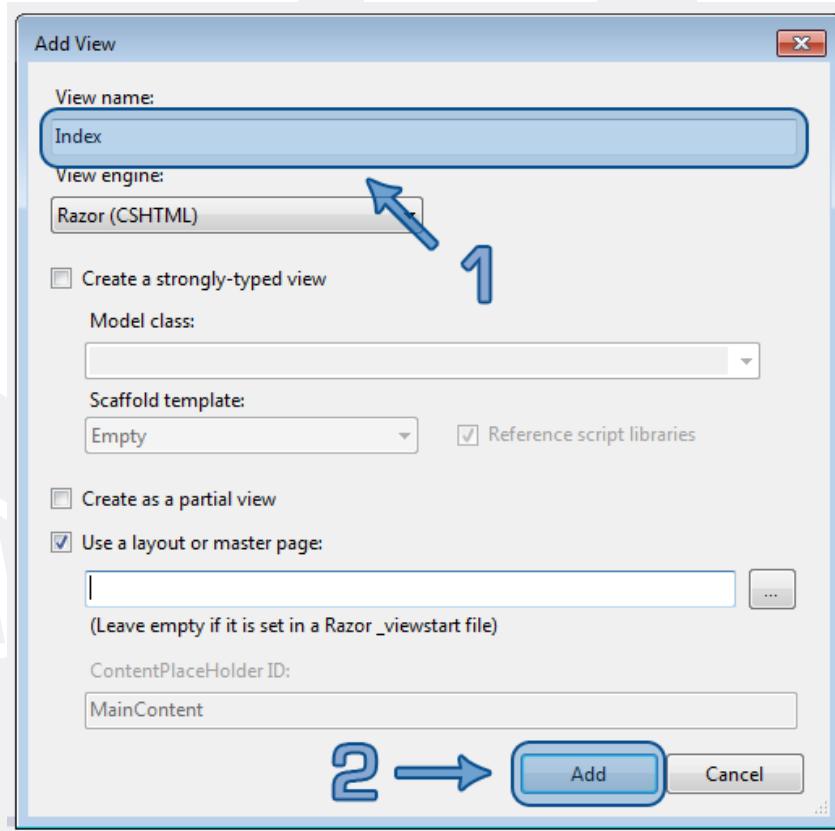


Figura 4.10: Adicionando uma tela

```

1 @{
2     ViewBag.Title = "Index";
3 }
4 <h2>Olá! O seu número da sorte é @ViewBag.NumeroDaSorte</h2>

```

Código CSHTML 4.2: Index.cshtml

- 5 Para visualizar a página, basta executar o projeto no Visual Studio e acessar o endereço
http://localhost:<PORTA_APP>/K19/Index



Figura 4.11: Acessando a página da aplicação

Integração com Entity Framework

A integração entre Entity Framework e ASP.NET MVC 4 é realizada de maneira extremamente simples. No Visual Studio 2012, ao criar um projeto ASP.NET MVC 4 do tipo *Basic*, a biblioteca do Entity Framework 5 é adicionada automaticamente.

As configurações padrões do Entity Framework podem ser alteradas através do arquivo **Web.config**.

Por fim, bastaria implementar as entidades e mapeá-las como visto no Capítulo 3.

Scaffold

O Visual Studio 2012 é capaz de gerar telas e controladores para as operações de inserção, leitura, alteração e remoção (CRUD) a partir de uma entidade de um projeto ASP.NET MVC 4. Os controladores gerados podem utilizar as funcionalidades do Entity Framework para interagir com o banco de dados.



Exercícios de Fixação

- 6 Vamos gerar o nosso primeiro scaffold. Para isto, defina as classes **Editora** e **K19Context** na pasta **Models** do projeto **K19**.

```

1 namespace K19.Models
2 {
3     public class Editora
4     {
5         public int EditoraID { get; set; }
6
7         public string Nome { get; set; }
8
9         public string Email { get; set; }
10    }
11 }
```

Código C# 4.3: *Editora.cs*

```

1 using System.Data.Entity;
2
3 namespace K19.Models
4 {
5     public class K19Context : DbContext
6     {
7         public DbSet<Editora> Editoras { get; set; }
8     }
9 }
```

Código C# 4.4: *K19Context.cs*

- 7 Antes de gerar o scaffold, é necessário fazer um build no projeto.

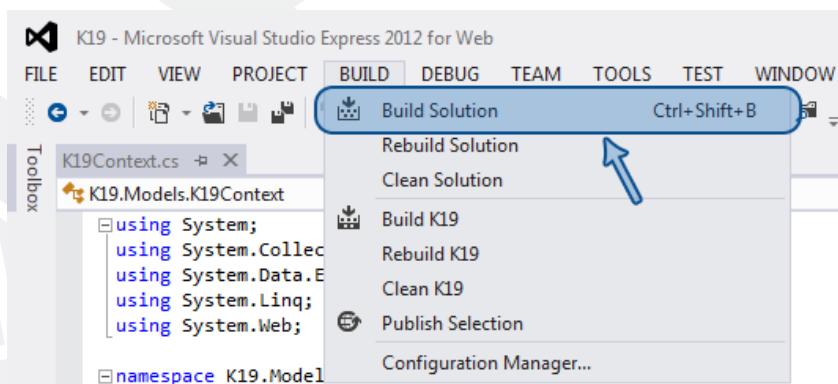


Figura 4.12: Build do projeto

- 8 Após o build do projeto, podemos gerar o scaffold. Para isso, crie um controlador chamado **Editora** seguindo os passos apresentados nas imagens a seguir.

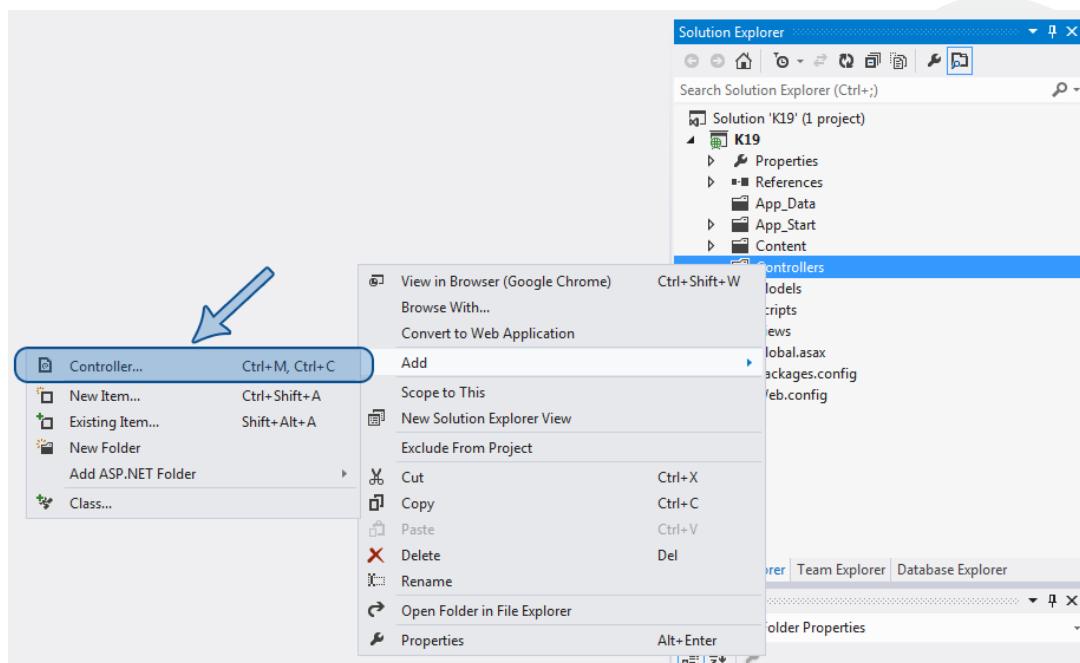


Figura 4.13: Gerando o scaffold

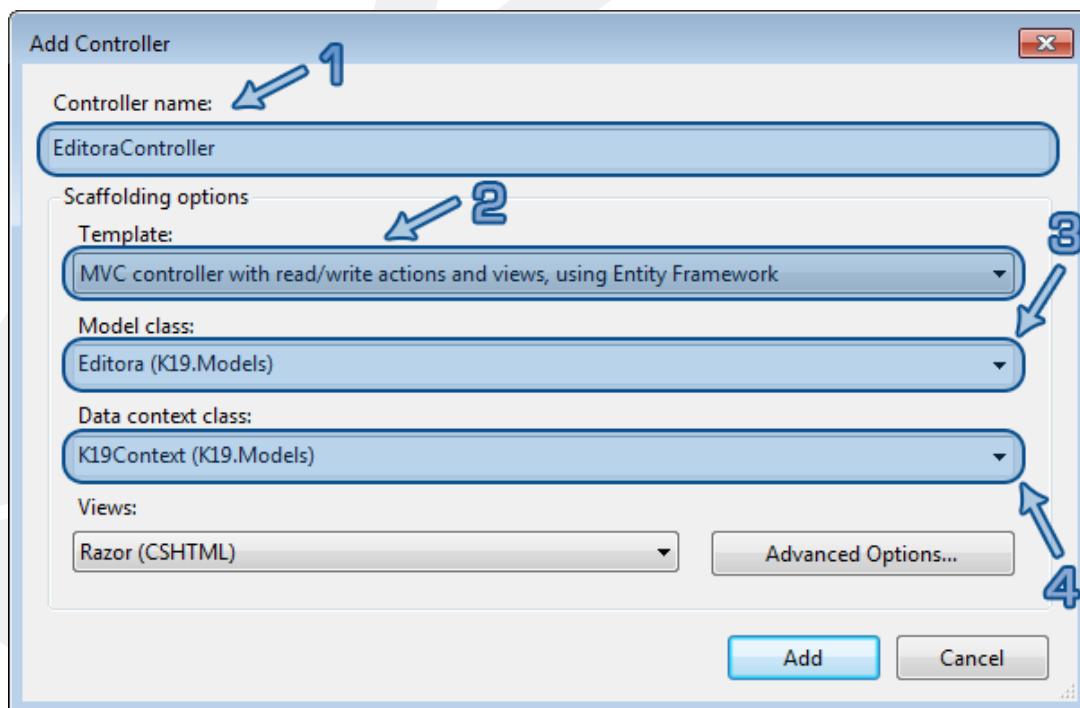


Figura 4.14: Gerando o scaffold

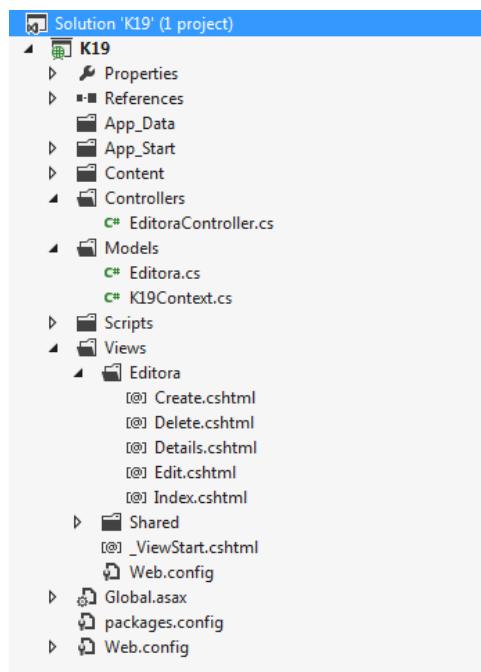


Figura 4.15: Scaffold gerado

- 9 Para testar, acesse o endereço http://localhost:<PORTA_APP>/Editora/Index.
- 10 Adicione uma entidade chamada **Livro** na pasta **Models**.

```

1 namespace K19.Models
2 {
3     public class Livro
4     {
5         public int LivroID { get; set; }
6
7         public string Titulo { get; set; }
8
9         public double Preco { get; set; }
10
11        public int EditoraID { get; set; }
12
13        public virtual Editora Editora { get; set; }
14    }
15 }
```

Código C# 4.5: Livro.cs

- 11 Altere a classe **K19Context** para registrar a entidade **Livro**.

```

1 namespace K19.Models
2 {
3     public class K19Context : DbContext
4     {
5         public DbSet<Editora> Editoras { get; set; }
6
7         public DbSet<Livro> Livros { get; set; }
8     }
9 }
```

Código C# 4.6: Livro.cs

12 Faça novamente o build do projeto. Depois, gere o scaffold da entidade **Livro**. Crie um controlador chamado **Livro** seguindo os passos apresentados nas imagens a seguir.

13 Para testar, acesse o endereço **http://localhost:<PORTA_APP>/Livro/Index**.

Apague a base de dados K19.Models.K19Context antes de testar a aplicação

CAMADA DE APRESENTAÇÃO

A camada de apresentação é responsável por gerar as páginas de uma aplicação web. Os dados apresentados em uma página web são definidos na camada de modelo. A camada de controle recupera os dados da camada de modelo e os envia para a camada de apresentação. Basicamente, a camada de apresentação definirá como esses dados serão apresentados para os usuários da aplicação.

O fluxo inverso também ocorre. Ou seja, a camada de apresentação obtém dados inseridos pelos usuários e os envia para a camada de controle que, por sua vez, usa esses dados para alterar a camada de modelo.

Neste capítulo, mostraremos como a camada de apresentação de uma aplicação web é construída com a utilização de recursos do ASP.NET MVC como *Inline Code*, *HTML Helper*, *Layout* e *Partial Views*.

Razor e ASPX

Até a segunda versão do ASP.NET MVC, as telas (ou páginas) de uma aplicação web eram desenvolvidas em ASPX. A partir da terceira versão desse framework, podemos usar a linguagem Razor para construir essas telas. A principal característica da Razor é ser concisa e simples, diminuindo o número de caracteres e as tradicionais tags de scripts do ASP.NET MVC (<% %>).

Nos exemplos abaixo, mostraremos algumas diferenças entre Razor e ASPX.

Criando Variáveis

Toda expressão em Razor começa com o caractere @. Para criarmos variáveis, precisamos declará-las dentro de um bloco Razor:

```
1 @{
2     string nome = "K19";
3     string telefoneDaK19 = "2387-3791";
4     string enderecoDaK19 = "Av. Brigadeiro Faria Lima";
5     int numeroDaK19 = 1571;
6 }
```

Código CSHTML 5.1: Criando variáveis em Razor

```
1 <%
2     string nome = "K19";
3     string telefoneDaK19 = "2387-3791";
4     string enderecoDaK19 = "Av. Brigadeiro Faria Lima";
5     int numeroDaK19 = 1571;
6 %>
```

Código ASPX 5.1: Criando variáveis em ASPX

Acessando variáveis

As variáveis podem ser facilmente acessadas.

```
1 <p>Telefone da K19: @telefoneDaK19</p>
```

Código CSHTML 5.2: Acessando uma variável

```
1 <p>Telefone da K19: <%= telefoneDaK19 %></p>
```

Código ASPX 5.2: Acessando uma variável

Condicionais (if e else)

Podemos utilizar os comandos de controle de fluxo if e else. Veja os exemplos abaixo.

```
1 @if(numero == numeroDaK19)
2 {
3     <p>Chegou na K19!</p>
4 }
5 else
6 {
7     <p>Continue andando.</p>
8 }
9 <!-- ou -->
10 @{
11     int numero = 463;
12     if(numero == numeroDaK19)
13     {
14         <p>Chegou na K19!</p>
15     }
16     else
17     {
18         <p>Continue andando.</p>
19     }
20 }
```

Código CSHTML 5.3: Utilizando if e else em Razor

```
1 <%
2 if(numero == numeroDaK19)
3 {
4 %>
5     <p>Chegou na K19!</p>
6 <%
7 }
8 else
9 {
10 %>
11     <p>Continue andando.</p>
12 <%
13 }
14 %>
```

Código CSHTML 5.4: Utilizando if e else em ASPX

Laços

Podemos criar laços utilizando o comando for.

```
1 @for(int i = 0; i < 5; i++)
2 {
```

```

3   <p>i = @i</p>
4 }
```

Código CSHTML 5.5: Criando um laço em Razor

```

1 <%
2 for(int i = 0; i < 5; i++)
3 {
4 %>
5   <p>i = <%= i %> </p>
6 <%
7 }
8 %>
```

Código ASPX 5.3: Criando um laço em ASPX

Texto e código

```

1 @if (x == "nome")
2 {
3   @: O nome da editora é @editora.Nome
4 }
```

Código CSHTML 5.6: Texto e código em Razor

```

1 <%
2 if (x == "nome")
3 {
4 %>
5   O nome da editora é <%= editora.Nome %>
6 <%
7 }
8 %>
```

Código ASPX 5.4: Texto e código em ASPX

Comentários

Podemos adicionar comentários nas páginas.

```

1 /* Comentário */
```

Código CSHTML 5.7: Comentários em Razor

```

1 <%-- Comentário --%>
```

Código ASPX 5.5: Comentários em ASPX



Exercícios de Fixação

- 1 Crie um projeto do tipo **ASP.NET MVC 4 Web Application** chamado **CamadaDeApresentacao** no **Visual Studio 2012 Express for Web**. Utilize o template **Basic Project**.

- 2** Adicione um controlador chamado **Home** no projeto **CamadaDeApresentacao** utilizando o template **Empty MVC Controller**. Nesse controlador, defina uma ação chamada **Index**.

```

1 namespace CamadaDeApresentacao.Controllers
2 {
3     public class HomeController : Controller
4     {
5         public ActionResult Index()
6         {
7             Random random = new Random();
8             ViewBag.NumeroDaSorte = random.Next();
9             return View();
10        }
11    }
12 }
```

Código C# 5.1: Home.cs

- 3** Adicione uma página associada à ação **Index** do controlador **Home** do projeto **CamadaDeApresentacao**.

```

1 @{
2     ViewBag.Title = "Index";
3 }
4
5 @for(int i = 0; i < 10; i++)
6 {
7     <h2>Olá! O seu número da sorte é @ViewBag.NumeroDaSorte</h2>
8 }
```

Código CSHTML 5.8: Index.cshtml

Para ver o resultado, acesse a url http://localhost:<PORTA_APP>/Home/Index.

- 4** Altere o arquivo **Index.cshtml** do exercício anterior para imprimir uma mensagem a partir de uma variável.

```

1 @{
2     ViewBag.Title = "Index";
3     string mensagem = "Olá! O seu número da sorte é " + ViewBag.NumeroDaSorte;
4 }
5
6 @for(int i = 0; i < 10; i++)
7 {
8     <h2>@mensagem</h2>
9 }
```

Código CSHTML 5.9: Index.cshtml

Para ver o resultado, acesse a url http://localhost:<PORTA_APP>/Home/Index.



Exercícios Complementares

- 1** Altere o arquivo **Index.cshtml** para gerar a seguinte tela para o usuário:

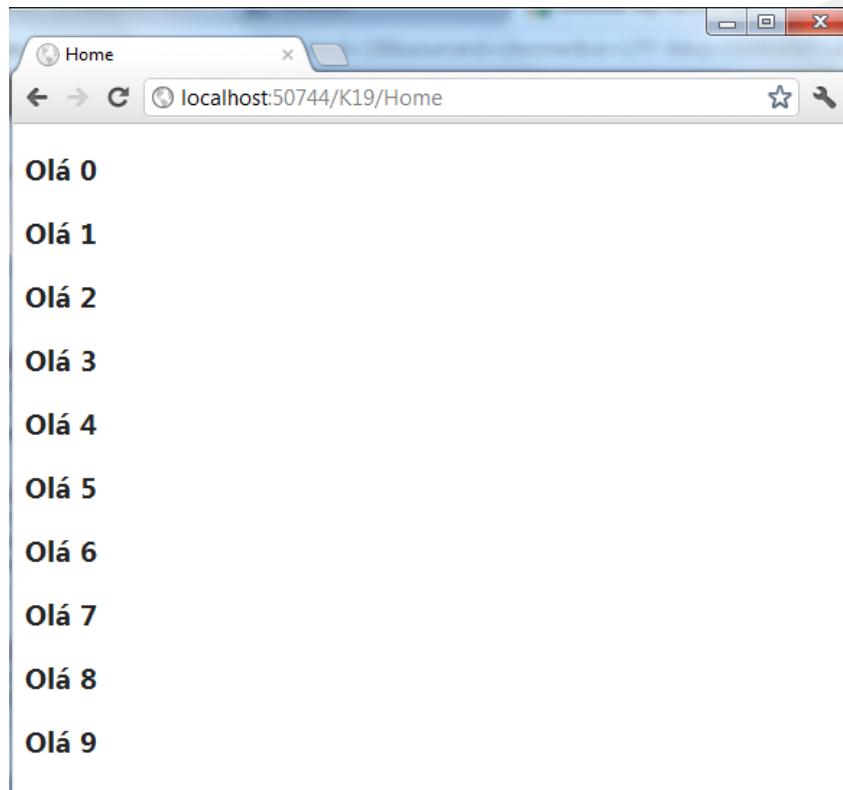


Figura 5.1: Inline Code

ViewBag e Strongly Typed Views

A ViewBag é utilizada para transmitir dados da camada de controle para a camada de apresentação. Para adicionar um item à propriedade ViewBag, devemos definir uma chave para esse item. Veja o código abaixo.

```
1 ViewBag.HoraDoServidor = DateTime.Now.ToShortTimeString();
```

Código C# 5.2: Adicionando um item na ViewBag

No código acima, criamos uma chave chamada HoraDoServidor e atribuímos um valor a essa chave.

Na camada de apresentação, os itens armazenados na ViewBag podem ser acessados facilmente através da sintaxe da Razor. Veja um exemplo abaixo.

```
1 A hora do servidor é @ViewBag.HoraDoServidor
```

Código CSHTML 5.11: Acessando um item da ViewBag

A ViewBag oferece uma forma simples de transmitir dados da camada de controle para a camada de apresentação, mas apresenta algumas desvantagens. Por exemplo, como as chaves são adicionadas dinamicamente, não é possível garantir a existência dessas chaves em tempo de compilação. Considere o código abaixo.

```

1 double numero = new Random().NextDouble();
2 if(numero < 0.5)
3 {
4     ViewBag.X = "K19 TREINAMENTOS";
5 }
6 else
7 {
8     ViewBag.X = 10;
9 }

```

Código C# 5.3: Adcionando chaves dinamicamente na ViewBag

No código acima, o tipo do objeto associado à chave X depende do valor da variável numero, que é gerado aleatoriamente. Quando o valor gerado for menor do que 0.5, o tipo desse objeto será string. Caso contrário, o tipo desse objeto será int. Suponha que a chave X seja utilizada na camada de apresentação.

```
1 <p>@ViewBag.X.ToLower()</p>
```

Código CSHTML 5.12: Utilizando a chave X ViewBag

O código acima não produz erros de compilação. Contudo, na execução, um erro poderá ocorrer. O problema é que esse código supõe que o objeto associado à chave X possui o método Lower(). No entanto, se a ação for acionada um número suficiente de vezes, em algum momento isso não será verdade, e um erro de execução ocorrerá.

Além disso, recursos do Visual Studio como *IntelliSense* (recurso para completar código) ou refatoração não podem ser aplicados com o uso da ViewBag.

Uma alternativa à utilização da ViewBag é o recurso **strongly typed views**. Com esse recurso podemos fixar, em tempo de compilação, o tipo do objeto que será transferido da camada de controle para a camada de apresentação. Uma vez que o tipo desse objeto foi fixado, o compilador é capaz de verificar a validade do código, evitando eventuais erros de execução. Além disso, o Visual Studio pode auxiliar o desenvolvimento da aplicação com os recursos de *IntelliSense* e de refatoração.

Para utilizar *strongly type views*, o primeiro passo é, na camada de controle, passar como parâmetro para o método View() um objeto do tipo esperado pela camada de apresentação.

```

1 public ActionResult Acao()
2 {
3     Editora editora = new Editora { Nome = "K19", Email = " contato@k19.com.br" };
4     return View(editora);
5 }

```

Código C# 5.4: Passando um objeto para a camada de apresentação

Para definir o tipo do objeto esperado na camada de apresentação, podemos utilizar a sintaxe da Razor. Veja o código abaixo.

```
1 @model K19.Models.Editora
```

Código CSHTML 5.13: Definindo o tipo do objeto esperado na camada de apresentação

Podemos também utilizar o comando using para evitar a escrita dos nomes completos das classes.

```

1 @using K19.Models
2 @model Editora

```

Código CSHTML 5.14: Utilizando o comando using

Para acessar o objeto transmitido da camada de controle para a camada de apresentação, devemos utilizar a propriedade Model. Observe o código a seguir.

```

1 @using K19.Models
2 @model Editora
3
4 <p>
5 Nome: @Model.Nome
6 <br />
7 Email: @Model.Email
8 </p>

```

Código CSHTML 5.15: Utilizando a propriedade Model

Se tentarmos acessar um método ou propriedade inexistentes, um erro de compilação ocorrerá, pois agora o compilador conhece o tipo do objeto transmitido. Dessa forma, o código abaixo produz um erro de compilação.

```

1 @using K19.Models
2 @model Editora
3
4 <p>
5 Nome: @Model.Nome
6 <br />
7 Telefone: @Model.Telefone
8 </p>

```

Código CSHTML 5.16: Produzindo um erro de compilação

Observe que apenas um objeto pode ser transmitido da camada de controle para a camada de apresentação através do recurso *strongly type views*. Já com a utilização de ViewBag, diversos objetos podem ser transmitidos. Note também que os dois recursos podem ser utilizados ao mesmo tempo.



Exercícios de Fixação

- 5 No projeto **CamadaDeApresentacao**, adicione um controlador chamado **Relogio** utilizando o template **Empty MVC Controller**. Nesse controlador, defina uma ação chamada **Agora**. Essa ação deve armazenar a data e o horário atual na ViewBag.

```

1 namespace CamadaDeApresentacao.Controllers
2 {
3     public class RelogioController : Controller
4     {
5         public ActionResult Agora()
6         {
7             ViewBag.Agora = DateTime.Now.ToString("dd/MM/yyyy HH:mm:ss");
8             return View();
9         }
10    }
11 }

```

Código C# 5.5: RelogioController.cs

- 6 Adicione uma página associada à ação **Agora** do controlador **Relogio**.

```
1 @{
2     ViewBag.Title = "Agora";
3 }
4 <h2>Agora: @ViewBag.Agora</h2>
```

Código HTML 5.1: Agora.cshtml

Para testar, acesse: http://localhost:<PORTA_APP>/Relogio/Agora.

- 7 Adicione uma entidade chamada **Aluno** na pasta **Models** do projeto **CamadaDeApresentacao**.

```
1 namespace CamadaDeApresentacao.Models
2 {
3     public class Aluno
4     {
5         public int AlunoID { get; set; }
6
7         public string Nome { get; set; }
8
9         public string Email { get; set; }
10    }
11 }
```

Código C# 5.6: Aluno.cs

- 8 Adicione um controlador chamado **Aluno** utilizando o template **Empty MVC Controller**. Nesse controlador, defina uma ação para transmitir para camada de apresentação um objeto da entidade **Aluno**. Utilize *strongly type views*.

```
1 namespace CamadaDeApresentacao.Controllers
2 {
3     public class AlunoController : Controller
4     {
5         public ActionResult Detalhes()
6         {
7             Aluno a = new Aluno {
8                 AlunoID = 1,
9                 Nome = "Jonas Hirata",
10                Email = "jonas@k19.com.br"
11            };
12            return View(a);
13        }
14    }
15 }
```

Código C# 5.7: AlunoController.cs

- 9 Adicione uma tela associada à ação **Detalhes** do controlador **Aluno** com o seguinte conteúdo.

```
1 @model CamadaDeApresentacao.Models.Aluno
```

```

2
3 @{
4     ViewBag.Title = "Detalhes";
5 }
6
7 <h2>Detalhes</h2>
8
9 <p>
10    AlunoID: @Model.AlunoID
11 </p>
12
13 <p>
14    Nome: @Model.Nome
15 </p>
16
17 <p>
18    Email: @Model.Email
19 </p>

```

Código HTML 5.2: Detalhes.cshtml

Para ver o resultado, acesse a url http://localhost:<PORTA_APP>/Aluno/Detalhes.

10 Adicione uma ação no controlador **Aluno**.

```

1 ...
2 public ActionResult Lista()
3 {
4     ICollection<Aluno> lista = new List<Aluno>();
5
6     for (int i = 0; i < 5; i++)
7     {
8         Aluno a = new Aluno {
9             AlunoID = i,
10            Nome = "Aluno " + i,
11            Email = "Email " + i
12        };
13        lista.Add(a);
14    }
15
16    return View(lista);
17 }
18 ...

```

Código C#5.8: AlunoController.cs

11 Adicione uma tela associada à ação **Lista** do controlador **Aluno** com o seguinte conteúdo.

```

1 @model ICollection<CamadaDeApresentacao.Models.Aluno>
2 @{
3     ViewBag.Title = "Lista";
4 }
5
6 <h2>Tabela de Aluno</h2>
7
8 <table>
9     <tr>
10        <th>AlunoID</th>
11        <th>Nome</th>
12        <th>Email</th>
13    </tr>
14
15    @foreach (var a in @Model)

```

```

17     {
18         <tr>
19             <td>@a.AlunoID</td>
20             <td>@a.Nome</td>
21             <td>@a.Email</td>
22     </tr>
23 }
24 </table>
```

Código HTML 5.3: Lista.cshtml

Para ver o resultado, acesse a url http://localhost:<PORTA_APP>/Aluno/Lista.

HTML Helpers

A função das páginas CSHTML é gerar hipertexto XHTML para enviar aos navegadores dos usuários. Os arquivos .cshtml misturam tags XHTML com scripts de servidor escritos em C# (ou outra linguagem de programação suportada pelo .NET). Essa mistura pode prejudicar a legibilidade do código em alguns casos. Além disso, a manutenção da aplicação pode se tornar mais complexa.

Considere a criação de um link utilizando diretamente as tags HTML.

```
1 <a href="/Editora/Index/">Lista de editoras</a>
```

Código CSHTML 5.17: Link para a lista de editoras

Se o formato da url para acessar a ação Index do controlador Editora for alterado, o código acima deverá ser modificado. Veremos como o formato das urls pode ser configurado no Capítulo ??.

Para facilitar a manutenção do código das páginas CSHTML, o ASP.NET MVC oferece os chamados **HTML Helpers**. A função de um HTML Helper é encapsular um código XHTML. Por exemplo, para adicionar um link, podemos usar o método **ActionLink** do objeto **Html**, ao invés de usar a tag <a> do HTML diretamente. No código abaixo, criamos um link para a ação responsável por listar as editoras.

```
1 @Html.ActionLink("Lista de editoras", "Index", "Editora")
```

Código CSHTML 5.18: Criando um link para a listagem de editoras

Agora, se o formato da url para acessar a ação Index do controlador Editora for alterado, o código acima não precisa ser alterado.

ActionLink Helper

O helper ActionLink é utilizado para gerar os links das páginas de uma aplicação web. Esse helper pode ser utilizado de diversas maneiras. A forma mais simples de utilizá-lo é passar a ele dois parâmetros: o texto e a ação associados ao link desejado. Nesse caso, o link gerado pelo helper ActionLink estará associado ao controlador correspondente a página na qual o link foi inserido.

```
1 @Html.ActionLink("TEXTO PARA O USUÁRIO", "ACTION")
```

Código CSHTML 5.19: Criando um link com controlador implícito

Podemos definir o controlador desejado explicitamente. Para isso, é necessário passar um terceiro parâmetro para o método ActionLink.

```
1 @Html.ActionLink("TEXTO PARA O USUÁRIO", "ACTION", "CONTROLADOR")
```

Código CSHTML 5.20: Criando um link com controlador explícito

O helper ActionLink permite que parâmetros sejam adicionados nos links gerados. Para isso, é necessário passar um array como parâmetro.

```
1 @Html.ActionLink("TEXTO PARA O USUÁRIO", "ACTION", new { Inicio = 0, Final = 10 })
```

Código CSHTML 5.21: Acrescenta parâmetros de url

BeginForm e EndForm Helpers

No ASP.NET MVC, há um conjunto de HTML Helpers para facilitar a criação de formulários nas páginas de uma aplicação web.

Para criar um formulário, utilizamos o helper BeginForm. Para terminar um formulário, utilizamos o helper EndForm. Veja o exemplo a seguir.

```
1 @{Html.BeginForm();}  
2  
3 <!-- elementos do formulário -->  
4  
5 @{Html.EndForm();}
```

Código CSHTML 5.22: Utilizando os helpers BeginForm e EndForm

Também podemos utilizar o comando using para garantir que os formulários sejam fechados. Nesse caso, não é necessário adicionar o helper EndForm. Observe o formulário abaixo.

```
1 @using(Html.BeginForm()) {  
2  
3 <!-- elementos do formulário -->  
4  
5 }
```

Código CSHTML 5.23: Utilizando o comando using

Por padrão, um formulário criado com o helper BeginForm enviará requisições à ação associada à url atual. Contudo, podemos definir explicitamente uma outra ação para receber essas requisições. Veja um exemplo no código abaixo.

```
1 @using(Html.BeginForm("ACTION", "CONTROLADOR")) {  
2  
3 <!-- elementos do formulário -->  
4  
5 }
```

Código CSHTML 5.24: Definindo uma ação e um controlador explicitamente

Por padrão, os formulários gerados pelo helper BeginForm fazem requisições do tipo POST. Nesse caso, as ações associadas a esses formulários devem ser anotadas com `HttpPost` para indicar o tipo de requisição HTTP esperado.

```

1 public class K19Controller : Controller
2 {
3     // POST: /K19/Home
4     [HttpPost]
5     public ActionResult Home()
6     {
7         ...
8         return View();
9     }
10 }
```

Código C# 5.9: Anotando uma ação com `HttpPost`

CheckBox

Podemos adicionar um checkbox em um formulário através do helper `CheckBox`.

```
1 @Html.CheckBox("Aceito", false)
```

Código CSHTML 5.25: Utilizando o helper `CheckBox`

O código acima produz o seguinte trecho de código HTML:

```

1 <input id="Aceito" name="Aceito" type="checkbox" value="true" />
2 <input name="Aceito" type="hidden" value="false" />
```

Código HTML 5.4: HTML gerado pelo helper `CheckBox`

Considere o formulário a seguir:

```

1 @using(Html.BeginForm("Cadastra", "Contrato"))
2 {
3     @Html.CheckBox("Aceito", false)
4     <input type="submit" value="Cadastra Contrato"/>
5 }
```

Código CSHTML 5.26: Formulário

Agora, considere a ação associada ao formulário criado no código acima.

```

1 public class ContratoController : Controller
2 {
3     // POST: /Contrato/Cadastra
4     [HttpPost]
5     public ActionResult Cadastra(Contrato contrato)
6     {
7         ...
8         return View();
9     }
10 }
```

Código C# 5.10: Ação

O valor enviado através do checkbox inserido no formulário será armazenado na propriedade `Aceito` do contrato enviado como parâmetro para a ação `Cadastra()`. Dessa forma, a classe `Contrato` deve possuir uma propriedade booleana chamada `Aceito`.

```

1 public class Contrato
2 {
```

```
3     public Boolean Aceito { get; set; }
4 }
```

Código C# 5.11: Contrato.cs

TextBox

Uma caixa de texto pode ser adicionada através do helper TextBox.

```
1 @Html.TextBox("Nome", "Digite o seu nome")
```

Código CSHTML 5.27: Utilizando o helper TextBox

O código acima produz o seguinte trecho de código HTML:

```
1 <input id="Nome" name="Nome" type="text" value="Digite o seu nome" />
```

Código HTML 5.5: HTML gerado pelo helper TextBox

TextArea

Uma caixa para textos maiores pode ser adicionada através do helper TextArea.

```
1 @Html.TextArea("Mensagem", "Digite uma mensagem")
```

Código CSHTML 5.28: Utilizando o helper TextArea

O código acima produz o seguinte trecho de código HTML:

```
1 <textarea cols="20" id="Mensagem" name="Mensagem" rows="2">Digite uma mensagem</textarea>
```

Código HTML 5.6: HTML gerado pelo helper TextArea

RadioButton

Um botão do tipo *radio* pode ser adicionado através do helper RadioButton.

```
1 @Html.RadioButton("CidadeNatal", "São Paulo", true)
2 @Html.RadioButton("CidadeNatal", "Natal", false)
3 @Html.RadioButton("CidadeNatal", "Piracicaba", false)
```

Código CSHTML 5.29: Utilizando o helper RadioButton

O código acima produz o seguinte trecho de código HTML:

```
1 <input checked="checked" id="CidadeNatal" name="CidadeNatal" type="radio" value="São Paulo" />
2 <input id="CidadeNatal" name="CidadeNatal" type="radio" value="Natal" />
3 <input id="CidadeNatal" name="CidadeNatal" type="radio" value="Piracicaba" />
```

Código HTML 5.7: HTML gerado pelo helper RadioButton

Hidden

Um campo escondido pode ser adicionado através do helper Hidden.

```
1 @Html.Hidden("Id", 1)
```

Código CSHTML 5.30: Utilizando o helper Hidden

O código acima produz o seguinte trecho de código HTML:

```
1 <input id="Id" name="Id" type="hidden" value="1" />
```

Código HTML 5.8: HTML gerado pelo helper Hidden

Password

Uma caixa para senha pode ser adicionada através do helper Password.

```
1 @Html.Password("Password", "senha")
```

Código CSHTML 5.31: Utilizando o helper Password

O código acima produz o seguinte trecho de código HTML:

```
1 <input id="Password" name="Password" type="password" value="senha" />
```

Código HTML 5.9: HTML gerado pelo helper Password



Mais Sobre

Os dados preenchidos nos elementos de um formulário são recuperados da ViewBag. Considere o código abaixo:

```
1 @Html.TextBox("Nome")
```

Código CSHTML 5.32: Utilizando o helper TextBox

Caso exista uma chave chamada Nome na ViewBag, então o valor associado a essa chave será o valor inicial da caixa de texto. Podemos alterar esse comportamento, passando um segundo parâmetro para o helper TextBox, que será o valor inicial da caixa de texto. Observe o código a seguir.

```
1 @Html.TextBox("Nome", "NomeInicial")
```

Código CSHTML 5.33: Utilizando o helper TextBox

Strongly Typed Helpers

Se os HTML Helpers forem utilizados de maneira análoga à mostrada anteriormente, a probabilidade de ocorrer um erro de digitação é alta. A forma que os HTML Helpers foram aplicados não permite que o compilador verifique a existência das propriedades associadas aos elementos dos formulários. Por exemplo, considere o código a seguir.

```
1 @Html.CheckBox("Aceito", false)
```

Código CSHTML 5.34: Utilizando o helper CheckBox

O código anterior supõe que o objeto recebido como parâmetro pela ação associada ao formulário onde o checkbox foi inserido possua uma propriedade chamada Aceito. Essa propriedade pode não existir. Contudo, nenhum erro de compilação seria gerado nesse caso.

Para evitar esse tipo de problema, podemos utilizar a seguinte sintaxe em telas fortemente tipadas:

```
1 @model K19.Models.Contrato
2 ...
3 @Html.CheckBoxFor(c => c.Aceito)
4 ...
```

Código CSHTML 5.35: Utilizando o helper CheckBoxFor

Com essa sintaxe, o compilador tem condições de verificar a existência das propriedades. Sendo assim, ele produzirá um erro de compilação se uma propriedade inexistente for utilizada.

TextBoxFor

```
1 @model K19.Models.Empresa
2 ...
3 @Html.TextBoxFor(x => x.Nome)
4 ...
```

Código CSHTML 5.36: Utilizando o helper TextBoxFor

O código acima produz o seguinte trecho de código HTML:

```
1 <input id="Nome" name="Nome" type="text" value="K19 Treinamentos" />
```

Código HTML 5.10: HTML gerado pelo helper TextBoxFor

TextAreaFor

```
1 @model K19.Models.Empresa
2 ...
3 @Html.TextAreaFor(x => x.Descricao)
4 ...
```

Código CSHTML 5.37: Utilizando o helper TextAreaFor

O código acima produz o seguinte trecho de código HTML:

```
1 <textarea cols="20" id="Descricao" name="Descricao" rows="2">
2   K19 Treinamentos em Java e .NET
3 </textarea>
```

Código HTML 5.11: HTML gerado pelo helper TextAreaFor

RadioButtonFor

```

1 @model K19.Models.Pessoa
2 ...
3 @Html.RadioButtonFor(x => x.CidadeNatal, "São Paulo")
4 @Html.RadioButtonFor(x => x.CidadeNatal, "Natal")
5 @Html.RadioButtonFor(x => x.CidadeNatal, "Piracicaba")
6 ...

```

Código CSHTML 5.38: Utilizando o helper RadioButtonFor

O código acima produz o seguinte trecho de código HTML:

```

1 <input checked="checked" id="CidadeNatal" name="CidadeNatal" type="radio" value="São ←
  Paulo" />
2 <input id="CidadeNatal" name="CidadeNatal" type="radio" value="Natal"/>
3 <input id="CidadeNatal" name="CidadeNatal" type="radio" value="Piracicaba"/>

```

Código HTML 5.12: HTML gerado pelo helper RadioButtonFor

HiddenFor

```

1 @model K19.Models.Empresa
2 ...
3 @Html.HiddenFor(e => e.Id)
4 ...

```

Código CSHTML 5.39: Utilizando o helper HiddenFor

O código acima produz o seguinte trecho de código HTML:

```

1 <input id="Id" name="Id" type="hidden" value="1" />

```

Código HTML 5.13: HTML gerado pelo helper HiddenFor

PasswordFor

```

1 @model K19.Models.Usuario
2 ...
3 @Html.PasswordFor(x => x.Senha)
4 ...

```

Código CSHTML 5.40: Utilizando o helper PasswordFor

O código acima produz o seguinte trecho de código HTML:

```

1 <input id="Senha" name="Senha" type="password" />

```

Código HTML 5.14: HTML gerado pelo helper PasswordFor



Mais Sobre

Os dados preenchidos nos elementos de um formulário de uma tela fortemente tipada são recuperados da propriedade Model. Considere o código abaixo:

```

1 @model K19.Models.Pessoa
2 ...
3 @Html.TextBoxFor(x => x.Nome)
4 ...

```

Código CSHTML 5.41: Utilizando o helper TextBoxFor

Caso exista a propriedade `Model.Nome`, o valor inicial dessa caixa de texto será o valor dessa propriedade.

DropDownList Helper

Considere uma aplicação para cadastrar livros de uma biblioteca. Quando um livro é cadastrado, o usuário pode escolher a editora desse livro. A editora pode ser selecionada através de um *drop down list*. Para criar um drop down list, podemos utilizar o HTML Helper `DropDownList.cshtml`.

O primeiro passo para utilizar o helper `DropDownList` é criar uma `SelectList` na camada de controle com as opções que o usuário poderá selecionar.

```
1 public class LivroController : Controller
2 {
3     public ActionResult Create()
4     {
5         List<Editora> editoras = editoraRepository.BuscaTodas();
6         ViewBag.Editoras = new SelectList(editoras, "Id", "Nome");
7         return View();
8     }
9 }
```

Código C# 5.12: LivroController.cs

O segundo passo é adicionar o helper `DropDownList` na camada de apresentação.

```
1 @Html.DropDownList("Editoras")
```

Código CSHTML 5.42: Utilizando o helper DropDownList

EditorFor

Considere a seguinte entidade.

```
1 public class Editora
2 {
3     public int Id { get; set; }
4     public string Nome { get; set; }
5     public string Email { get; set; }
6     public bool IsAtivo { get; set; }
7     public string Descricao { get; set; }
8     public virtual ICollection<Livro> Livros { get; set; }
9 }
```

Código C# 5.13: Editora.cs

Para editar os dados de uma editora, temos uma página conforme o exemplo abaixo:

```
1 @model K19.Models.Editora
2 @{
3     ViewBag.Title = "Edição de Editora";
4 }
```

```

7 <h2>Edição de Editora</h2>
8
9 @using (Html.BeginForm()) {
10   @Html.HiddenFor(model => model.Id)
11
12   @Html.LabelFor(model => model.Nome)
13   @Html.TextBoxFor(model => model.Nome)
14
15   <br />
16
17   @Html.LabelFor(model => model.Email)
18   @Html.TextBoxFor(model => model.Email)
19
20   <br />
21
22   @Html.LabelFor(model => model.Descricao)
23   @Html.TextAreaFor(model => model.Descricao)
24
25   <br />
26
27   @Html.LabelFor(model => model.IsAtivo)
28   @Html.CheckBoxFor(model => model.IsAtivo)
29
30   <input type="submit" value="Salvar" />
31 }
```

Código CSHTML 5.43: Edit.cshtml

Para cada propriedade da entidade `Editora`, utilizamos um helper para gerar o código HTML necessário para a entrada de dados. Por exemplo, no caso das propriedades `Nome` e `Email`, utilizamos o Helper `TextBox`. Para a propriedade booleana `IsAtivo`, utilizamos o helper `CheckBox`.

Observe que a escolha do helper depende basicamente do tipo da propriedade associada a ele. Podemos deixar essa escolha a cargo do helper `EditorFor`. Para propriedades de tipo booleano, esse helper utilizará o `CheckBox`. Para propriedades do tipo `string`, esse helper utilizará o `TextBox`. Observe a utilização do helper `EditorFor` no código abaixo.

```

1 @model K19.Models.Editora
2
3 @{
4   ViewBag.Title = "Edição de Editora";
5 }
6
7 <h2>Edição de Editora</h2>
8
9 @using (Html.BeginForm()) {
10   @Html.HiddenFor(model => model.Id)
11
12   @Html.LabelFor(model => model.Nome)
13   @Html.EditorFor(model => model.Nome)
14
15   <br />
16
17   @Html.LabelFor(model => model.Email)
18   @Html.EditorFor(model => model.Email)
19
20   <br />
21
22   @Html.LabelFor(model => model.Descricao)
23   @Html.EditorFor(model => model.Descricao)
24
25   <br />
26
27   @Html.LabelFor(model => model.IsAtivo)
28   @Html.EditorFor(model => model.IsAtivo)
29 
```

```
30 <input type="submit" value="Salvar" />
31 }
```

Código CSHTML 5.44: Edit.cshtml

EditorForModel

Também temos o helper `EditorForModel` para construir um formulário completo com base nas propriedades de uma entidade. Esse helper seguirá a mesma abordagem do `EditorFor`. Ou seja, para cada propriedade do Model, ele utilizará o helper apropriado.

```
1 @model K19.Models.Editora
2
3 @{
4     ViewBag.Title = "Edição de Editora";
5 }
6
7 <h2>Edição de Editora</h2>
8
9 @using (Html.BeginForm()) {
10     @Html.EditorForModel()
11
12     <input type="submit" value="Save" />
13 }
```

Código CSHTML 5.45: Edit.cshtml

Personalizando o EditorFor e o EditorForModel

Para personalizar o funcionamento do helper `EditorFor` e do `EditorForModel`, podemos utilizar algumas anotações. Veja o exemplo abaixo.

```
1 public class Usuario
2 {
3     [HiddenInput(DisplayValue = false)]
4     public int Id { get; set; }
5
6     [Display(Name = "Nome do Usuário")]
7     public string Nome { get; set; }
8
9     [DataType(DataType.EmailAddress)]
10    public string Email { get; set; }
11
12    [DataType(DataType.Password)]
13    public string Senha { get; set; }
14
15    [DataType(DataType.MultilineText)]
16    public string Descricao { get; set; }
17
18    [DataType(DataType.Date)]
19    [DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:yyyy-MM-dd}")]
20    public DateTime DataDeCadastro { get; set; }
21 }
```

Código C# 5.14: Usuario.cs



Exercícios de Fixação

- 12 Adicione uma ação no controlador **Aluno** com o seguinte código.

```

1 ...
2 public ActionResult Editar()
3 {
4     Aluno aluno = new Aluno
5     {
6         AlunoID = 1,
7         Nome = "Jonas Hirata",
8         Email = "jonas@k19.com.br"
9     };
10    return View(aluno);
11 }
12 ...

```

Código C# 5.15: AlunoController.cs

- 13 Adicione uma página associada à ação **Editar** do controlador **Aluno**.

```

1 @model CamadaDeApresentacao.Models.Aluno
2 @{
3     ViewBag.Title = "Editar";
4 }
5
6 <h2>Editar</h2>
7
8 <form action="/Aluno/Editar" method="post">
9     <input id="AlunoID" name="AlunoID" type="hidden" value="1" />
10
11     <label for="Nome">Nome</label>
12     <input id="Nome" name="Nome" type="text" value="@Model.Nome" />
13
14     <label for="Email">E-mail</label>
15     <input id="Email" name="Email" type="text" value="@Model.Email" />
16
17     <input type="submit" value="Salvar" />
18 </form>

```

Código CSHTML 5.46: Editar.cshtml

Para testar, acesse a url http://localhost:<PORTA_APP>/Aluno/Editar.

- 14 Altere o exercício anterior, porém utilizando HTML Helpers.

```

1 @model CamadaDeApresentacao.Models.Aluno
2 @{
3     ViewBag.Title = "Editar";
4 }
5
6 <h2>Editar</h2>
7
8 @using (Html.BeginForm())
{
9     @Html.Hidden("AlunoID")
10    @Html.Label("Nome")
11    @Html.TextBox("Nome")
12    @Html.Label("Email")
13    @Html.TextBox("Email")
14    <input type="submit" value="Salvar" />
15 }

```

Código CSHTML 5.47: Editar.cshtml

Para testar, acesse a url http://localhost:<PORTA_APP>/Aluno/Editar.

- 15** Altere o exercício anterior e utilize strongly typed helpers.

```

1 @model CamadaDeApresentacao.Models.Aluno
2 @{
3     ViewBag.Title = "Editar";
4 }
5
6 <h2>Editar</h2>
7
8 @using (Html.BeginForm())
{
9
10    @Html.HiddenFor(x => x.AlunoID)
11    @Html.LabelFor(x => x.Nome)
12    @Html.TextBoxFor(x => x.Nome)
13    @Html.LabelFor(x => x.Email)
14    @Html.TextBoxFor(x => x.Email)
15    <input type="submit" value="Salvar" />
16 }
```

Código CSHTML 5.48: Editar.cshtml

Para testar, acesse a url http://localhost:<PORTA_APP>/Aluno/Editar.

- 16** Altere o exercício anterior e utilize o HTML Helper EditorFor.

```

1 @model CamadaDeApresentacao.Models.Aluno
2 @{
3     ViewBag.Title = "Editar";
4 }
5
6 <h2>Editar</h2>
7
8 @using (Html.BeginForm())
{
9
10    @Html.EditorFor(x => x.AlunoID)
11    @Html.LabelFor(x => x.Nome)
12    @Html.EditorFor(x => x.Nome)
13    @Html.LabelFor(x => x.Email)
14    @Html.EditorFor(x => x.Email)
15    <input type="submit" value="Salvar" />
16 }
```

Código HTML 5.15: Editar.cshtml

Para testar, acesse a url http://localhost:<PORTA_APP>/Aluno/Editar.

OBS: repare que o campo correspondente à propriedade AlunoID é exibido no formulário.

- 17** Altere a entidade Aluno para indicar que a propriedade AlunoID não deve ser exibida nos formulários que utilizarem o HTML Helper EditorFor e EditorForModel.

```

1 namespace CamadaDeApresentacao.Models
2 {
3     public class Aluno
4     {
5         [HiddenInput(DisplayValue = false)]
6         public int AlunoID { get; set; }
7 }
```

```

8     public string Nome { get; set; }
9
10    public string Email { get; set; }
11
12 }
```

Código C# 5.16: Aluno.cs

Para testar, acesse a url http://localhost:<PORTA_APP>/Aluno/Editar.

OBS: repare que o campo correspondente à propriedade AlunoID não é exibido no formulário.

- 18 Altere o exercício anterior e utilize o HTML Helper EditorForModel.

```

1 @model CamadaDeApresentacao.Models.Aluno
2 @{
3     ViewBag.Title = "Editar";
4 }
5
6 <h2>Editar</h2>
7
8 @using (Html.BeginForm())
9 {
10     @Html.EditorForModel()
11     <input type="submit" value="Salvar" />
12 }
```

Código HTML 5.16: Editar.cshtml

Para testar, acesse a url http://localhost:<PORTA_APP>/Aluno/Editar.

- 19 Adicione uma nova entidade chamada **Usuario** na pasta **Models** do projeto **CamadaDeApresentacao**.

```

1 namespace CamadaDeApresentacao.Models
2 {
3     public class Usuario
4     {
5         [HiddenInput(DisplayValue = false)]
6         public int Id { get; set; }
7
8         [Display(Name = "Nome do Usuário")]
9         public string Nome { get; set; }
10
11        [DataType(DataType.EmailAddress)]
12        public string Email { get; set; }
13
14        [DataType(DataType.Password)]
15        public string Senha { get; set; }
16
17        [DataType(DataType.MultilineText)]
18        public string Descricao { get; set; }
19
20        [DataType(DataType.Date)]
21        [DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:yyyy-MM-dd}")]
22        public DateTime DataDeCadastro { get; set; }
23    }
24 }
```

Código C# 5.17: Usuario.cs

- 20** Adicione um novo controlador chamado **Usuario**.

```

1 namespace CamadaDeApresentacao.Controllers
2 {
3     public class UsuarioController : Controller
4     {
5         public ActionResult Editar()
6         {
7             Usuario usuario = new Usuario
8             {
9                 Id = 1,
10                Nome = "Rafael Cosentino",
11                Email = "rafael@k19.com.br",
12                Senha = "123",
13                Descricao = "Sócio-fundador da K19 / Instrutor",
14                DataDeCadastro = DateTime.Now
15            };
16            return View(usuario);
17        }
18    }
19 }
```

Código C# 5.18: *UsuarioController.cs*

- 21** Adicione uma página associada à ação **Editar** do controlador **Usuario**.

```

1 @model CamadaDeApresentacao.Models.Usuario
2 @{
3     ViewBag.Title = "Editar";
4 }
5 <h2>Editar</h2>
6 @using (Html.BeginForm())
7 {
8     @Html.EditorForModel()
9     <input type="submit" value="Salvar" />
10 }
```

Código CSHTML 5.49: *Editar.cshtml*

Layouts

É comum que as páginas de uma aplicação web possuam conteúdo em comum (por exemplo, um cabeçalho ou um rodapé). O conteúdo em comum pode ser replicado em todas as páginas através do CTRL+C e CTRL+V. Porém, essa não é uma boa abordagem, pois quando alguma alteração precisa ser realizada, todos os arquivos devem ser modificados. Também é comum que as páginas de uma aplicação web possuam um certo padrão visual. Daí surge o conceito de **Layouts**.

Conteúdo comum

Tudo que é comum a todas as páginas de um determinado grupo pode ser definido em um Layout. Dessa forma, qualquer alteração é facilmente realizada modificando-se apenas um arquivo. Por exemplo, suponha que toda página de uma aplicação web deva ter o mesmo título e a mesma formatação. Podemos criar um Layout com o título desejado e com a referência ao arquivo CSS que define a formatação padrão.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>@ViewBag.Title</title>
5   @Styles.Render("~/Content/Site.css")
6   @Scripts.Render("~/Scripts/jquery-1.5.1.min.js")
7 </head>
8
9 <body>
10  <div id="header">
11    @Html.ActionLink("Editoras", "Index", "Editora")
12    @Html.ActionLink("Livros", "Index", "Livro")
13  </div>
14  @RenderBody()
15 </body>
16 </html>

```

Código CSHTML 5.50: K19Layout.cshtml

No layout definido no código acima, utilizamos o método `Render()` das propriedades `Styles` e `Scripts` para adicionar conteúdo CSS e JavaScript. Na tag `<title>`, acrescentamos `@ViewBag.Title`, para que cada página possa definir o seu próprio título. Por fim, o método `RenderBody()` indica onde o conteúdo definido em cada página será adicionado.

É recomendado que os arquivos que definam layouts sejam colocados na pasta `Views\Share`.



Mais Sobre

Suponha que devemos adicionar diversos arquivos CSS à uma página de uma aplicação ASP.NET MVC. Se esses arquivos forem adicionados um a um da maneira tradicional (com a tag `<link>`), o número de requisições que os navegadores terão de realizar para carregar essa página será alto, afetando o desempenho. Para contornar esse problema, poderíamos agrupar o código CSS de todos os arquivos em apenas um. Contudo, essa abordagem dificultaria a manutenção do código CSS. Dessa forma, o ASP.NET MVC 4 possui um mecanismo capaz de agrupar o conteúdo CSS de diversos arquivos dinamicamente. Para usar esse mecanismo, basta invocar o método `Render()` da propriedade `Styles`.

```

1  @Styles.Render("~/Content/Arquivo1.css", "~/Content/Arquivo2.css", "~/Content/Arquivo3.css")

```

Código CSHTML 5.51: Utilizando o método Render para adicionar conteúdo CSS

Analogamente, a propriedade `Scripts` possui o método `Render()`, que também tem por objetivo agrupar o código JavaScript de diversos arquivos.

Com o layout definido, o próximo passo é indicar quais páginas utilizarão esse layout. Por exemplo, podemos atualizar a página de edição de editoras para utilizar `K19Layout.cshtml` como layout.

```

1 @model K19.Models.Editora
2
3 @{
4   Layout = "~/Views/Shared/K19Layout.cshtml";
5
6   // Define o título específico desta página
7   ViewBag.Title = "Edição de Editora";
8 }
9
10 <h2>Edição de Editora</h2>

```

```

11 @using(Html.BeginForm()) {
12     @Html.EditorForModel()
13     <input type="submit" value="Salvar" />
14 }
15

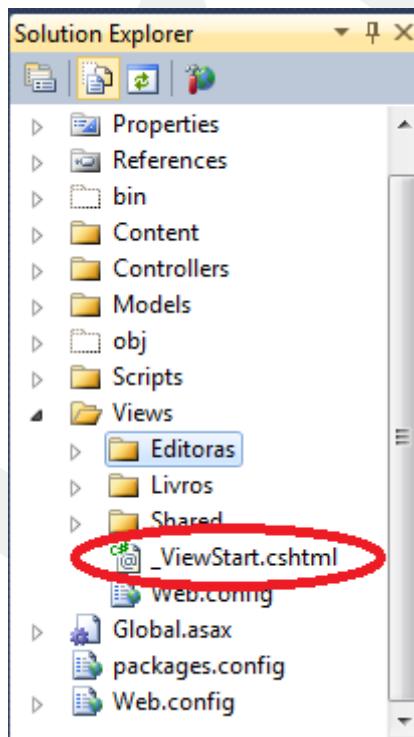
```

Código CSHTML 5.52: Edit.cshtml

Quando a página de edição de editoras é requisitada, o arquivo `Edit.cshtml` é processado antes do arquivo `K19Layout.cshtml`, o que permite definir o valor de `ViewBag.Title` no arquivo `Edit.cshtml` e utilizá-lo no layout.

Para definir o layout de `Edit.cshtml`, foi necessário armazenar o caminho completo do arquivo que define o layout na propriedade `Layout`. Este procedimento não é muito prático, pois em cada página que deseja utilizar esse layout é necessário definir esta propriedade.

A partir da terceira versão do ASP.NET MVC, temos uma nova funcionalidade que permite definir um layout padrão para todas as páginas, não havendo necessidade de definir a propriedade `Layout` em cada página. Para isso, basta acrescentarmos o arquivo `_ViewStart.cshtml` à pasta `View`:



O `_ViewStart.cshtml` permite definirmos um código que será executado antes de cada página ser renderizada. Nesse arquivo podemos definir, por exemplo, a propriedade `Layout`:

```

1 @{
2     Layout = "~/Views/Shared/K19Layout.cshtml";
3 }

```

Código CSHTML 5.53: _ViewStart.cshtml

Como este código é processado antes dos arquivos específicos de cada página, não há mais necessidade de definir a propriedade `Layout` em cada arquivo específico.

Lacunas

Também podemos criar seções em um layout para serem preenchidas com conteúdos específicos definidos nas páginas. Considere o seguinte layout.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>@ViewBag.Title</title>
5   @Styles.Render("~/Content/Site.css")
6   @Scripts.Render("~/Scripts/jquery-1.5.1.min.js")
7 </head>
8
9 <body>
10  <div id="header">
11    @Html.ActionLink("Editoras", "Index", "Editora")
12    @Html.ActionLink("Livros", "Index", "Livro")
13  </div>
14  <div id="sidebar">@RenderSection("Sidebar", false)</div>
15  <div id="content">@RenderBody()</div>
16  <div id="footer">K19 Treinamentos</div>
17 </body>
18 </html>
```

Código CSHTML 5.54: K19Layout.cshtml

No código acima, utilizamos o método `@RenderSection()` para criar uma seção no layout. O primeiro parâmetro ("Sidebar") é o nome da seção e o segundo é um booleano que indica se o preenchimento dessa seção é obrigatório ou não.

Para definir o conteúdo de uma seção, devemos utilizar o código `@section`. Observe o código de uma página que utiliza o layout criado anteriormente e define a seção Sidebar.

```

1 @model K19.Models.Editora
2 @{
3   Layout = "~/Views/Shared/K19Layout.cshtml";
4
5   // Define o título específico desta página
6   ViewBag.Title = "Edição de Editora";
7 }
8
9 <h2>Edição de Editora</h2>
10
11 @using(Html.BeginForm()) {
12   @Html.EditorForModel()
13   <input type="submit" value="Salvar" />
14 }
15
16
17 @section Sidebar {
18   <p>Sidebar do cadastro de Edição de Editora</p>
19 }
```

Código CSHTML 5.55: Edit.cshtml



Exercícios de Fixação

- 22** Crie um arquivo chamado **K19Layout.cshtml** dentro da pasta **Shared** que fica na pasta **Views** do projeto **CamadaDeApresentacao**. Este arquivo servirá de modelo para as páginas da nossa aplicação.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width" />
6      <title>@ViewBag.Title</title>
7      @Styles.Render("~/Content/css")
8      @Scripts.Render("~/bundles/modernizr")
9  </head>
10 <body>
11     <header>
12         
14         <span>K19 Treinamentos</span>
15     </header>
16
17     <section id="main">
18         @RenderBody()
19     </section>
20
21     <footer>&copy;K19 Treinamentos</footer>
22     @Scripts.Render("~/bundles/jquery")
23     @RenderSection("scripts", required: false)
24 </body>
25 </html>

```

Código CSHTML 5.56: K19Layout.cshtml

- 23** Altere a página **Lista.cshtml** da pasta **Views/Aluno** para que ela utilize o layout definido no exercício anterior.

```

1  @model ICollection<CamadaDeApresentacao.Models.Aluno>
2
3  @{
4      ViewBag.Title = "Lista";
5      Layout = "~/Views/Shared/K19Layout.cshtml";
6  }
7
8  <h2>Tabela de Aluno</h2>
9
10 <table>
11     <tr>
12         <th>AlunoID</th>
13         <th>Nome</th>
14         <th>Email</th>
15     </tr>
16
17     @foreach (var a in @Model)
18     {
19         <tr>
20             <td>@a.AlunoID</td>
21             <td>@a.Nome</td>
22             <td>@a.Email</td>
23         </tr>
24     }
25 </table>

```

Código HTML 5.17: Lista.cshtml

Teste acessando o endereço: http://localhost:<PORTA_APP>/Aluno/Lista.

- 24** Defina a página **K19Layout.cshtml** como layout padrão para todas as telas. Altere o arquivo **_ViewStart.cshtml** da pasta **Views**.

```
1 @{
2     Layout = "~/Views/Shared/K19Layout.cshtml";
3 }
```

Código CSHTML 5.57: _ViewStart.cshtml

Teste acessando as páginas da aplicação.

- http://localhost:<PORTA_APP>/Home/Index
- http://localhost:<PORTA_APP>/Aluno/Detalhes
- http://localhost:<PORTA_APP>/Aluno/Editar
- http://localhost:<PORTA_APP>/Aluno/Lista
- http://localhost:<PORTA_APP>/Usuario/Editar
- http://localhost:<PORTA_APP>/Relogio/Agora

- 25 Altere o arquivo **Site.css** da pasta **Content** para aplicar algumas regras de formatação às nossas páginas.

```
1 ...
2 header,
3 #main,
4 footer {
5     width: 980px;
6     margin: 0 auto;
7 }
8
9 header {
10    border-bottom: 1px solid #666666;
11    padding: 0 0 10px 0;
12 }
13
14 header span {
15    font-size: 20px;
16    vertical-align: middle;
17 }
18
19 header img {
20    height: 80px;
21    margin: 0 20px 0 0;
22    vertical-align: middle;
23 }
24
25 footer {
26    border-top: 1px solid #666666;
27    padding: 10px 0 0 0;
28    margin-top: 20px;
29    text-align: center;
30    font-size: 10px;
31 }
32
33 #toolbar {
34    background: #999999;
35    width: 970px;
36    margin: 1px auto;
37    padding: 3px 5px;
38 }
39
40 #toolbar * {
41    margin: 3px 5px;
42 }
```

Código CSS 5.1: Site.css

Teste acessando as páginas da aplicação (Dica: utilize o comando CTRL+F5 para garantir que o navegador solicite o novo conteúdo do arquivo Site.css).

- `http://localhost:<PORTA_APP>/Home/Index`
- `http://localhost:<PORTA_APP>/Aluno/Detalhes`
- `http://localhost:<PORTA_APP>/Aluno/Editar`
- `http://localhost:<PORTA_APP>/Aluno/Lista`
- `http://localhost:<PORTA_APP>/Usuario/Editar`
- `http://localhost:<PORTA_APP>/Relogio/Agora`

26 Altere o arquivo **K19Layout.cshtml** da pasta **Views\Shared**.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <meta name="viewport" content="width=device-width" />
6     <title>@ViewBag.Title</title>
7     @Styles.Render("~/Content/css")
8     @Scripts.Render("~/bundles/modernizr")
9 </head>
10 <body>
11     <header>
12         
14         <span>K19 Treinamentos</span>
15     </header>
16     <div id="toolbar">
17         @RenderSection("Toolbar", required: false)
18     </div>
19
20     <section id="main">
21         @RenderBody()
22     </section>
23
24     <footer>&copy;K19 Treinamentos</footer>
25     @Scripts.Render("~/bundles/jquery")
26     @RenderSection("scripts", required: false)
27 </body>
28 </html>
```

Código CSHTML 5.58: K19Layout.cshtml

27 Defina a seção **Toolbar** nas seguintes páginas.

```

1 @model CamadaDeApresentacao.Models.Aluno
2 @{
3     ViewBag.Title = "Editar";
4 }
5
6 @section Toolbar {
7     <input type="button" value="Remover este cadastro" />
8 }
```

```

10 <h2>Editar</h2>
11 @using (Html.BeginForm())
12 {
13     @Html.EditorForModel()
14     <input type="submit" value="Enviar" />
15 }
16

```

Código CSHTML 5.59: Editar.cshtml

```

1 @model CamadaDeApresentacao.Models.Aluno
2 @{
3     ViewBag.Title = "Detalhes";
4 }
5
6 @section Toolbar {
7     <input type="button" value="Remover este cadastro" />
8 }
9
10
11 <h2>Detalhes</h2>
12
13 <p>
14     AlunoID: @Model.AlunoID
15 </p>
16
17 <p>
18     Nome: @Model.Nome
19 </p>
20
21 <p>
22     Email: @Model.Email
23 </p>

```

Código CSHTML 5.60: Detalhes.cshtml

```

1 @model ICollection<CamadaDeApresentacao.Models.Aluno>
2 @{
3     ViewBag.Title = "Lista";
4     Layout = "~/Views/Shared/K19Layout.cshtml";
5 }
6
7 @section Toolbar {
8     <input type="search" placeholder="Digite o nome do aluno" />
9     <input type="button" value="Buscar" />
10 }
11
12
13 <h2>Tabela de Aluno</h2>
14
15 <table>
16     <tr>
17         <th>AlunoID</th>
18         <th>Nome</th>
19         <th>Email</th>
20     </tr>
21
22     @foreach (var a in @Model)
23     {
24         <tr>
25             <td>@a.AlunoID</td>
26             <td>@a.Nome</td>
27             <td>@a.Email</td>
28         </tr>
29     }
30 </table>

```

Código CSHTML 5.61: Lista.cshtml

Teste acessando as páginas da aplicação.

- http://localhost:<PORTA_APP>/Home/Index
- http://localhost:<PORTA_APP>/Aluno/Detalhes
- http://localhost:<PORTA_APP>/Aluno/Editar
- http://localhost:<PORTA_APP>/Aluno/Lista
- http://localhost:<PORTA_APP>/Usuario/Editar
- http://localhost:<PORTA_APP>/Relogio/Agora

28 Defina um conteúdo padrão para a seção **Toolbar** no layout.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <meta name="viewport" content="width=device-width" />
6     <title>@ViewBag.Title</title>
7     @Styles.Render("~/Content/css")
8     @Scripts.Render("~/bundles/modernizr")
9 </head>
10 <body>
11     <header>
12         
14         <span>K19 Treinamentos</span>
15     </header>
16     <div id="toolbar">
17         @if (IsSectionDefined("Toolbar"))
18         {
19             @RenderSection("Toolbar", required: false);
20         }
21         else
22         {
23             <input type="button"
24                 value="Veja o seu número da sorte"
25                 onclick="location.href='/Home/Index'" />
26         }
27     </div>
28
29     <section id="main">
30         @RenderBody()
31     </section>
32
33     <footer>&copy;K19 Treinamentos</footer>
34     @Scripts.Render("~/bundles/jquery")
35     @RenderSection("scripts", required: false)
36 </body>
37 </html>
```

Código CSHTML 5.62: K19Layout.cshtml

Teste acessando as páginas da aplicação.

- http://localhost:<PORTA_APP>/Home/Index
- http://localhost:<PORTA_APP>/Aluno/Detalhes
- http://localhost:<PORTA_APP>/Aluno/Editar
- http://localhost:<PORTA_APP>/Aluno/Lista

- http://localhost:<PORTA_APP>/Usuario/Editar
- http://localhost:<PORTA_APP>/Relogio/Agora

Partial views

Quanto mais elaborada é uma página de uma aplicação web, mais extenso é o seu código. Códigos muito extensos prejudicam a legibilidade e a manutenção da aplicação. Para organizar melhor o código, podemos dividir o conteúdo de uma página web em vários arquivos.

Suponha que desejamos dividir o conteúdo de uma página em duas partes. Devemos criar um arquivo para cada parte.

```
1 <h1> Parte 1 </h1>
2 <p> Conteúdo da parte1</p>
```

Código CSHTML 5.63: *_Parte1.cshtml*

```
1 <h1> Parte 2 </h1>
2 <p> Conteúdo da parte2</p>
```

Código CSHTML 5.64: *_Parte2.cshtml*



Mais Sobre

Por convenção, o nome dos arquivos que definem as telas parciais de uma aplicação ASP.NET MVC devem iniciar com o caractere “_”.

Por fim, devemos criar um arquivo principal para agrupar essas partes. Utilizaremos o método `Partial()` para inserir o conteúdo dos arquivos secundários no arquivo principal.

```
1 <html>
2   <head>
3     <title>Exemplo de partial</title>
4   </head>
5
6   <body>
7     Html.Partial("_Parte1")
8     Html.Partial("_Parte2")
9   </body>
10 </html>
```

Código CSHTML 5.65: *Principal.cshtml*

O método `Partial()` procura os arquivos `_Parte1.cshtml` e `_Parte2.cshtml` no mesmo diretório do arquivo principal. Caso ele não encontre, ele continua procurando esses arquivos na pasta `Views\Shared`. Essa lógica também é aplicada no método `View` que utilizamos na camada de controle.

O recurso de páginas parciais permite a criação de conteúdo reutilizável de forma mais clara e concisa. Informações podem ser compartilhadas entre as páginas principais e as páginas parciais através da propriedade `ViewBag`. Por exemplo, podemos definir duas páginas principais, uma para

criar e outra para editar editoras. Nessas duas páginas podemos acrescentar uma página parcial com o formulário que pode ser utilizado para cadastrar novas editoras ou editar editoras existentes.

Veja abaixo o exemplo da página parcial que contém o formulário de cadastro ou edição de editoras.

```

1 <!-- ~/Views/Editora/_Form.cshtml -->
2 @model K19.Models.Editora
3
4 @Scripts.Render(
5     "~/Scripts/jquery-1.5.1.min.js",
6     "~/Scripts/jquery.validate.min.js",
7     "~/Scripts/jquery.validate.unobtrusive.min.js")
8
9 @using (Html.BeginForm()) {
10     @Html.ValidationSummary(true)
11     <fieldset>
12         <legend>Editora</legend>
13         @if (Model != null)
14         {
15             @Html.EditorFor(model => model.Id)
16         }
17
18         <div class="editor-label">
19             @Html.LabelFor(model => model.Nome)
20         </div>
21         <div class="editor-field">
22             @Html.EditorFor(model => model.Nome)
23         </div>
24
25         <div class="editor-label">
26             @Html.LabelFor(model => model.Email)
27         </div>
28         <div class="editor-field">
29             @Html.EditorFor(model => model.Email)
30         </div>
31
32         <div class="editor-label">
33             @Html.LabelFor(model => model.IsAtivo)
34         </div>
35         <div class="editor-field">
36             @Html.EditorFor(model => model.IsAtivo)
37         </div>
38
39         <p>
40             <input type="submit" value="Save" />
41         </p>
42     </fieldset>
43 }
```

Código CSHTML 5.66: _Form.cshtml

Veja abaixo o exemplo das páginas principais que utilizam a página parcial definida no código acima.

```

1 <!-- ~/Views/Editora/Create.cshtml -->
2 @model K19.Models.Editora
3
4 @{
5     ViewBag.Title = "Cadastro de Editora";
6 }
7
8 <h2>Cadastro de Editora</h2>
9
10 @Html.Partial("_Form")
```

Código CSHTML 5.67: Create.cshtml

```

1 <!-- ~/Views/Editora/Edit.cshtml -->
2 @model LivrariaVirtual.Models.Editora
3
4 @{
5     ViewBag.Title = "Edição de Editora";
6 }
7
8 <h2>Edição de Editora</h2>
9
10 @Html.Partial("_Form")

```

Código CSHTML 5.68: Edit.cshtml



Exercícios de Fixação

- 29 Adicione uma nova entidade chamada **Post** na pasta **Models**.

```

1 namespace CamadaDeApresentacao.Models
2 {
3     public class Post
4     {
5         public string Autor { get; set; }
6
7         public DateTime Data { get; set; }
8
9         public string Titulo { get; set; }
10
11        public string Categoria { get; set; }
12
13        public string Texto { get; set; }
14    }
15 }

```

Código C# 5.19: Post.cs

- 30 Adicione um novo controlador chamado **Post** utilizando o template **Empty MVC Controller**.

```

1 namespace CamadaDeApresentacao.Controllers
2 {
3     public class PostController : Controller
4     {
5         public ActionResult Lista()
6         {
7             ICollection<Post> posts = new List<Post>();
8
9             for (int i = 0; i < 10; i++)
10             {
11                 posts.Add(new Post
12                 {
13                     Autor = "Autor do post " + i,
14                     Categoria = "Categoria " + i,
15                     Data = DateTime.Now.Subtract(TimeSpan.FromDays(i)),
16                     Texto = i + " - Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
17                     Titulo = "Título do post " + i
18                 });
19             }
}

```

```
20     return View(posts);
21 }
22 }
23 }
```

Código C# 5.20: PostController.cs

- 31 Adicione uma página associada à ação **Lista** do controlador **Post**.

```
1 @model IEnumerable<CamadaDeApresentacao.Models.Post>
2 @{
3     ViewBag.Title = "Lista";
4 }
5 <h2>Lista</h2>
6
7 @foreach (var post in Model) {
8     @Html.Partial("_Post", post)
9 }
```

Código CSHTML 5.69: Lista.cshtml

- 32 Adicione uma partial para exibir as informações de um Post. Crie um arquivo chamado **_Post.cshtml** na pasta **Views/Post**

```
1 @model CamadaDeApresentacao.Models.Post
2
3 <article>
4     <header>
5         <h1>@Model.Titulo</h1>
6         <div><small>@Model.Autor - @Model.Data.ToString("dd/MM/yyyy")</small></div>
7     </header>
8     <p>@Model.Texto</p>
9     <footer>
10        Categoria: @Model.Categoria
11    </footer>
12 </article>
```

Código CSHTML 5.70: _Post.cshtml

Para testar, acesse a página http://localhost:<PORTA_APP>/Post/Lista



CAMADA DE CONTROLE

No ASP.NET MVC as urls são mapeadas para métodos (ações) em classes que definem os chamadas controladores. As requisições enviadas pelos navegadores são processadas pelos controladores. O processamento realizado por um controlador para tratar uma requisição consiste basicamente em:

- Recuperar os dados enviados pelo usuário através de formulários.
- Interagir com a camada de modelo.
- Acionar a camada de apresentação para construir a página HTML que deve ser enviada para o usuário como resposta à sua requisição.

Para que uma classe seja considerada um controlador, ela deve seguir algumas regras.

- O nome da classe deve ter o sufixo “Controller”.
- A classe deve implementar a interface `System.Web.Mvc.IController` ou herdar da classe `System.Web.Mvc.Controller`.

Raramente, você definirá uma classe para criar um controlador implementando a interface `IController`. Comumente as classes que definem controladores derivam de `Controller`. Diversas propriedade e métodos são herdados da classe `Controller`. Essas propriedades facilitam o processamento das requisições e a utilização dos recursos do ASP.NET MVC.

Actions

Os controladores e as ações são elementos fundamentais de uma aplicação ASP.NET MVC. Um controlador pode conter diversas ações. As ações são utilizadas para processar as requisições realizadas pelos navegadores. Para criar uma ação, é necessário definir um método `public` dentro da classe de um controlador. Os parâmetros desse método podem ser utilizados para receber os dados enviados pelos usuários através de formulários HTML. Esse método deve devolver um `ActionResult` que será utilizado pelo ASP.NET MVC para definir o que deve ser executado depois que a ação terminar.

Quando um usuário faz uma requisição HTTP através de um navegador, o ASP.NET MVC verifica na tabela de rotas o controlador e a ação associados à url da requisição realizada. Essa tabela é definida no arquivo `RouteConfig.cs` e inicializada no arquivo `Global.asax`.

Por padrão, quando criamos um projeto ASP.NET MVC no Visual Studio, uma rota com o formato `{controller}/{action}/{id}` é adicionada na tabela de rotas. Com essa rota, se uma requisição for

realizada para a url `http://www.k19.com.br/Editora/Listagem`, o controlador definido pela classe `EditoraController` e a ação implementada pelo método `Listagem()` dessa classe serão escolhidos para processar essa requisição.

Se uma requisição for realizada para a url `http://www.k19.com.br/Editora/Remove/1`, o controlador definido pela classe `EditoraController` e a ação implementada pelo método `Remove()` dessa classe serão escolhidos para processar a requisição realizada. Além disso, o valor 1 será passado como parâmetro para o método `Remove()`. Veremos mais sobre rotas no Capítulo ??.

ActionResult

Quando uma ação termina, o método correspondente deve devolver um `ActionResult`. O valor devolvido indica para o ASP.NET MVC o que deve ser executado depois da ação. Veja abaixo uma lista com alguns tipos específicos de `ActionResult` que podem ser utilizados.

ViewResult: Devolve uma página da camada de apresentação. Considere o seguinte exemplo.

```
1 public class TesteController
2 {
3     public ActionResult Acao()
4     {
5         return View();
6     }
7 }
```

Considerando uma aplicação ASP.NET MVC 4 em C# e Razor, o método `View()`, ao ser chamado sem parâmetros, executará o seguinte processo para determinar qual arquivo deve ser utilizado para construir a página de resposta.

1. Se o arquivo `Views\teste\Acao.cshtml` existir, ele será utilizado.
2. Caso contrário, se o arquivo `Views\Shared\Acao.cshtml` existir, ele será utilizado.
3. Se nenhum desses arquivos existir, uma página de erro será devolvida.

Por outro lado, podemos especificar o nome do arquivo que define a página de resposta. Veja o exemplo abaixo.

```
1 return View("NomeDaView");
```

Além disso, podemos passar um objeto para a camada de apresentação. Esse objeto será utilizado nas páginas fortemente tipadas como vimos no Capítulo 5.

```
1 return View(editora);
```

Também é possível especificar o nome do arquivo que define a página de resposta e o objeto que deve ser transmitido para a camada de apresentação ao mesmo tempo.

```
1 return View("NomeDaView", editora);
```

PartialViewResult: Devolve uma página parcial da camada de apresentação. Exemplos:

```
1 return PartialView();
```

O método `PartialView()` utiliza a mesma abordagem do método `View()` para determinar o arquivo que deve ser utilizado para construir a página parcial de resposta.

```
1 return PartialView("NomeDaPartialView", editora);
```

RedirectResult: Redireciona o navegador para uma URL específica. Exemplo:

```
1 return Redirect("http://www.k19.com.br");
```

RedirectToAction: Redireciona para outra ação da camada de controle. Exemplo:

```
1 return RedirectToAction("OutraAction", "OutroController");
```

ContentResult: Devolve texto. Exemplo:

```
1 return Content("Texto", "text/plain");
```

JsonResult: Devolve um objeto no formato JSON. Exemplo:

```
1 return Json(editora);
```

JavaScriptResult: Devolve código Javascript. Exemplo:

```
1 return JavaScript("\$( '#divResultText' ).html('JavaScript Passed');");
```

FileResult: Devolve dados binários. Exemplo:

```
1 return File(@"c:\relatorio.pdf", "application/pdf");
```

EmptyResult: Devolve uma resposta vazia. Exemplo:

```
1 return new EmptyResult();
```

Parâmetros

Os parâmetros enviados pelos usuários através de formulários HTML podem ser recuperados por meio da propriedade `Request`.

```
1 string nome = Request["Nome"];
```

Código C# 6.14: Recuperando o parâmetro Nome enviado em uma requisição HTTP

Esses parâmetros também podem ser recuperados através dos parâmetros da ação responsável pelo processamento da requisição HTTP realizada. Para isso, basta definir um parâmetro C# para cada parâmetro HTTP com o mesmo nome. Veja o exemplo abaixo:

```

1 <html>
2 <head>
3   <title>Cadastro de Editora</title>
4 </head>
5 <body>
6   <form action="/Editoras/Salva" method="post">
7     Nome: <input type="text" name="nome"/>
8     Email: <input type="text" name="email"/>
9     <input type="submit" />
10  </form>
11 </body>
12 </html>

```

Código CSHTML 6.1: Cadastra.cshtml

No código acima, criamos um formulário HTML com os parâmetros nome e email. Esses parâmetros serão recuperados na ação Salva definida no código abaixo.

```

1 ...
2 [HttpPost]
3 public ActionResult Salva(string nome, string email)
4 {
5   Editora editora = new Editora { Nome = nome, Email = email };
6   db.Editoras.Add(editora);
7   return View();
8 }
9 ...

```

Código C# 6.15: EditoraController.cs

O ASP.NET MVC também é capaz de montar objetos com os valores dos parâmetros HTTP enviados pelo usuário e passá-los como argumento para as ações dos controladores.

```

1 [HttpPost]
2 public ActionResult Salva(Editora editora)
3 {
4   db.Editoras.Add(editora);
5   return View();
6 }

```

Código C# 6.16: EditoraController.cs

As propriedades dos objetos recebidos como argumentos nas ações dos controladores precisam ter os mesmos nomes dos parâmetros HTTP ignorando-se letras maiúsculas ou minúsculas.



Exercícios de Fixação

1 Crie um projeto do tipo **ASP.NET MVC 4 Web Application** chamado **CamadaDeControle** no **Visual Studio 2012 Express for Web**. Utilize o template **Basic Project**.

2 Adicione uma entidade chamada **Produto** na pasta **Models** do projeto **CamadaDeControle**.

```

1 namespace CamadaDeControle.Models
2 {
3   public class Produto
4   {

```

```

5   [HiddenInput(DisplayValue=false)]
6   public int ProdutoID { get; set; }
7
8   [Display(Name="Nome do Produto")]
9   public string Nome { get; set; }
10
11  [DataType(DataType.MultilineText)]
12  [Display(Name="Descrição do Produto")]
13  public string Descricao { get; set; }
14
15  [Display(Name="Preço do Produto")]
16  public double Preco { get; set; }
17
18 }
```

Código C# 6.17: Produto.cs

- 3 Adicione um DbContext para registrar a entidade **Produto** e utilizar os recursos do Entity Framework. Crie uma classe chamada **K19Context** na pasta **Models**.

```

1 namespace CamadaDeControle.Models
2 {
3     public class K19Context : DbContext
4     {
5         public DbSet<Produto> Produtos { get; set; }
6     }
7 }
```

Código C# 6.18: K19Context.cs

- 4 Adicione um controlador chamado **Produto** utilizando o template **Empty MVC Controller**.

```

1 namespace CamadaDeControle.Controllers
2 {
3     public class ProdutoController : Controller
4     {
5         public ActionResult Lista()
6         {
7             K19Context ctx = new K19Context();
8             return View(ctx.Produtos);
9         }
10
11        [HttpGet]
12        public ActionResult Cadastra()
13        {
14            return View();
15        }
16
17        [HttpPost]
18        public ActionResult Cadastra(string nome, string descricao, double preco)
19        {
20            Produto p = new Produto
21            {
22                Nome = nome,
23                Descricao = descricao,
24                Preco = preco
25            };
26
27            K19Context ctx = new K19Context();
28            ctx.Produtos.Add(p);
29            ctx.SaveChanges();
30
31            return RedirectToAction("Lista");
32        }
33 }
```

```
33     }
34 }
```

Código C# 6.19: ProdutoController.cs

- 5 Adicione uma tela associada à ação **Lista** do controlador **Produto** com o seguinte conteúdo.

```
1 @using CamadaDeControle.Models
2 @model IEnumerable<Produto>
3
4 @{
5     ViewBag.Title = "Lista";
6 }
7
8 <h2>Lista</h2>
9
10 <table>
11     <tr>
12         <th>@Html.DisplayNameFor(m => m.Nome)</th>
13         <th>@Html.DisplayNameFor(m => m.Descricao)</th>
14         <th>@Html.DisplayNameFor(m => m.Preco)</th>
15     </tr>
16     @foreach(var produto in Model)
17     {
18         <tr>
19             <td>@Html.DisplayFor(m => produto.Nome)</td>
20             <td>@Html.DisplayFor(m => produto.Descricao)</td>
21             <td>@Html.DisplayFor(m => produto.Preco)</td>
22         </tr>
23     }
24 </table>
```

Código HTML 6.1: Lista.cshtml

- 6 Adicione uma tela associada à ação **Cadastra** do controlador **Produto** com o seguinte conteúdo.

```
1 @model CamadaDeControle.Models.Produto
2
3 @{
4     ViewBag.Title = "Cadastra";
5 }
6
7 <h2>Cadastra</h2>
8
9 @using (Html.BeginForm())
{
10     @Html.EditorFor(m => m.ProdutoID)
11     <div>@Html.LabelFor(m => m.Nome)</div>
12     @Html.EditorFor(m => m.Nome)
13     <div>@Html.LabelFor(m => m.Descricao)</div>
14     @Html.EditorFor(m => m.Descricao)
15     <div>@Html.LabelFor(m => m.Preco)</div>
16     @Html.EditorFor(m => m.Preco)
17
18     <div><input type="submit" value="Cadastrar" /></div>
19
20 }
```

Código HTML 6.2: Cadastra.cshtml

Para ver o resultado, acesse a url http://localhost:<PORTA_APP>/Produto/Cadastra.

7 Altere a ação **Cadastra** do controlador **Produto**.

```

1  namespace CamadaDeControle.Controllers
2  {
3      public class ProdutoController : Controller
4      {
5          public ActionResult Lista()
6          {
7              K19Context ctx = new K19Context();
8              return View(ctx.Produtos);
9          }
10
11         [HttpGet]
12         public ActionResult Cadastra()
13         {
14             return View();
15         }
16
17         [HttpPost]
18         public ActionResult Cadastra(Produto p)
19         {
20             K19Context ctx = new K19Context();
21             ctx.Produtos.Add(p);
22             ctx.SaveChanges();
23
24             return RedirectToAction("Lista");
25         }
26     }
27 }
```

Código C# 6.20: ProdutoController.cs

Para testar, acesse a url http://localhost:<PORTA_APP>/Produto/Cadastra.

TempData

Ao efetuar um redirecionamento, uma nova requisição é realizada pelo navegador. Nesta nova requisição, não temos mais acesso aos dados e objetos da requisição anterior ao redirecionamento. Caso haja a necessidade de preservar dados ao longo do redirecionamento, podemos utilizar a propriedade **TempData**.

Por exemplo, ao cadastrar uma editora, podemos efetuar um redirecionamento para a tela de listagem de editoras. Na propriedade **TempData**, podemos acrescentar uma mensagem indicando o eventual sucesso da operação. Veja o exemplo abaixo.

```

1 ...
2 [HttpPost]
3 public ActionResult Salva(Editora editora)
4 {
5     db.Editoras.Add(editora);
6     db.SaveChanges();
7     TempData["Mensagem"] = "Editora cadastrada com sucesso!";
8     return RedirectToAction("Index");
9 }
10 ...
```

Código C# 6.21: EditoraController.cs

Na camada de apresentação, podemos recuperar os dados armazenados na propriedade **TempData**. Veja o código a seguir:

```

1 @if ( TempData["Mensagem"] != null)
2 {
3     <p>@TempData["Mensagem"]</p>
4 }

```

Código CSHTML 6.2: Recuperando dados armazenados na propriedade TempData



Exercícios de Fixação

- 8 Altere a ação **Cadastra** do controlador **Produto**.

```

1 ...
2 public ActionResult Cadastra(Produto p)
3 {
4     K19Context ctx = new K19Context();
5     ctx.Produtos.Add(p);
6     ctx.SaveChanges();
7
8     ViewBag.Mensagem = "Produto cadastrado com sucesso!";
9
10    return RedirectToAction("Lista");
11 }
12 ...

```

Código C# 6.22: ProdutoController.cs

- 9 Modifique o arquivo **Lista.cshtml** da pasta **Views/Produto**.

```

1 @using CamadaDeControle.Models
2 @model IEnumerable<Produto>
3
4 @{
5     ViewBag.Title = "Lista";
6 }
7
8 <h2>Lista</h2>
9
10 @if (ViewBag.Mensagem != null)
11 {
12     <div>@ViewBag.Mensagem</div>
13 }
14
15 <table>
16     <tr>
17         <th>@Html.DisplayNameFor(m => m.Nome)</th>
18         <th>@Html.DisplayNameFor(m => m.Descricao)</th>
19         <th>@Html.DisplayNameFor(m => m.Preco)</th>
20     </tr>
21     @foreach(var produto in Model)
22     {
23         <tr>
24             <td>@Html.DisplayFor(m => produto.Nome)</td>
25             <td>@Html.DisplayFor(m => produto.Descricao)</td>
26             <td>@Html.DisplayFor(m => produto.Preco)</td>
27         </tr>
28     }
29 </table>

```

Código CSHTML 6.3: Lista.cshtml

Teste a aplicação acessando a url http://localhost:<PORTA_APP>/Produto/Cadastra. Verifique que a mensagem de sucesso **não** aparecerá.

- 10** Altere novamente a ação **Cadastra** do controlador **Produto**.

```

1 ...
2 public ActionResult Cadastra(Produto p)
3 {
4     K19Context ctx = new K19Context();
5     ctx.Produtos.Add(p);
6     ctx.SaveChanges();
7
8     TempData["Mensagem"] = "Produto cadastrado com sucesso!";
9
10    return RedirectToAction("Lista");
11 }
12 ...

```

Código C# 6.23: ProdutoController.cs

- 11** Modifique o arquivo **Lista.cshtml** da pasta **Views/Produto**.

```

1 @using CamadaDeControle.Models
2 @model IEnumerable<Produto>
3
4 @{
5     ViewBag.Title = "Lista";
6 }
7
8 <h2>Lista</h2>
9
10 @if (TempData["Mensagem"] != null)
11 {
12     <div>@TempData["Mensagem"]</div>
13 }
14
15 <table>
16     <tr>
17         <th>@Html.DisplayNameFor(m => m.Nome)</th>
18         <th>@Html.DisplayNameFor(m => m.Descricao)</th>
19         <th>@Html.DisplayNameFor(m => m.Preco)</th>
20     </tr>
21     @foreach(var produto in Model)
22     {
23         <tr>
24             <td>@Html.DisplayFor(m => produto.Nome)</td>
25             <td>@Html.DisplayFor(m => produto.Descricao)</td>
26             <td>@Html.DisplayFor(m => produto.Preco)</td>
27         </tr>
28     }
29 </table>

```

Código CSHTML 6.4: Lista.cshtml

Teste a aplicação acessando a url http://localhost:<PORTA_APP>/Produto/Cadastra. Verifique que a mensagem de sucesso aparecerá.

Rotas

Para acessar uma determinada ação da nossa aplicação, os usuários devem realizar uma requisição HTTP utilizando a url correspondente à ação. Por exemplo, para acessar a listagem de editoras, é necessário digitar na barra de endereço do navegador a url `http://localhost:<PORTA_APP>/Editora/Index`. Perceba que o nome do controlador e o nome da ação desejados fazem parte da url. O formato dessas urls é definido por **rotas** criadas no arquivo `RouteConfig.cs` e carregadas no arquivo `Global.asax`. O código abaixo mostra a rota padrão inserida nos projetos ASP.NET MVC 4.

```
1 routes.MapRoute(
2     name: "Default",
3     url: "{controller}/{action}/{id}",
4     defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
5 );
```

Código C# 6.24: `RouteConfig.cs`

O primeiro argumento do método `MapRoute` é o nome da rota, o segundo é a expressão que define o formato da rota e o terceiro é o conjunto de valores padrão dos parâmetros da rota.

A expressão que determina o formato da rota `Default` utiliza três parâmetros: `controller`, `action` e `id`. Dessa forma, se o usuário realizar uma requisição HTTP utilizando url `http://localhost:<PORTA_APP>/Editora/Remove/1`, o ASP.NET MVC criará uma instância do controlador `Editora` e executará o método `Remove()` passando o valor `1` como argumento.

Basicamente, as rotas associam urls e ações. Veja alguns exemplos de como as urls serão processadas de acordo com as regras de rotas.

URL	Mapeamento da URL
/	<code>controller = "Home", action = "Index"</code>
/Livro	<code>controller = "Livro", action = "Index"</code>
/Livro/Adiciona	<code>controller = "Livro", action = "Adiciona"</code>
/Livro/Remove/1	<code>controller = "Livro", action = "Remove", id = 1</code>

Adicionando uma rota

Para acrescentar uma rota, podemos utilizar o método `MapRoute()`. Suponha que a listagem de editoras deva ser acessada através da url `http://localhost:<PORTA_APP>/Catalogo`. Nesse caso, podemos adicionar uma rota no arquivo `RoutesConfig.cs`, como mostrado abaixo.

```
1 ...
2 routes.MapRoute(
3     name: "Nova Rota",
4     url: "Catalogo",
5     defaults: new { controller = "Editora", action = "Index" }
6 );
7 ...
```

Código C# 6.25: `RoutesConfig.cs`



Importante

As rotas são processadas na ordem em que foram inseridas. Portanto, é importante definir as rotas mais específicas antes das rotas mais genéricas.

Adicionando Parâmetros nas Rotas

Podemos acrescentar parâmetros às rotas. Por exemplo, na rota criada anteriormente para a listagem de editoras, poderíamos adicionar um parâmetro para limitar a quantidade de editoras listadas.

```

1 routes.MapRoute(
2     name: "Nova Rota",
3     url: "Catalogo/{maximo}",
4     defaults: new { controller = "Editora", action = "Index" }
5 );

```

Código C# 6.26: RoutesConfig.cs

Por padrão, os parâmetros adicionados a uma rota são obrigatórios. Dessa forma, no exemplo acima, a listagem de editoras só poderá ser acessada se o valor do parâmetro `maximo` for definido na url da requisição. Por exemplo, http://localhost:<PORTA_APP>/Catalogo/20. Se uma requisição for realizada para a url http://localhost:<PORTA_APP>/Catalogo, um erro ocorrerá. Para mudar esse comportamento, podemos tornar o parâmetro `maximo` opcional. Veja o código a seguir.

```

1 ...
2 routes.MapRoute(
3     name: "Nova Rota",
4     url: "Catalogo/{maximo}",
5     defaults: new { controller = "Editora", action = "Index",
6         maximo = UrlParameter.Optional }
7 );
8 ...

```

Código C# 6.27: RoutesConfig.cs

Os parâmetros das rotas podem ser recuperados nas ações. Observe o exemplo abaixo.

```

1 ...
2 public ActionResult Index(int? maximo)
3 {
4     // implementação
5 }
6 ...

```

Código C# 6.28: EditoraController.cs

Ao definir parâmetros opcionais, devemos utilizar parâmetros do tipo `nullable type` nas ações, pois quando o parâmetro não estiver definido na url de uma requisição, o valor `null` será atribuído ao parâmetro da ação. Por exemplo, para os tipos `int` e `double`, devemos utilizar `int?` e `double?`, respectivamente.



Exercícios de Fixação

- 12** Altere a rota padrão do projeto **CamadaDeControle**. Para isso, modifique o arquivo **RouteConfig.cs** da pasta **App_Start**.

```

1 ...
2 routes.MapRoute(
3     name: "Default",

```

```

4     url: "{controller}/{action}/{id}",
5     defaults: new {
6         controller = "Produto",
7         action = "Lista",
8         id = UrlParameter.Optional
9     }
10    );
11 ...

```

Código C# 6.29: RouteConfig.cs

Teste essa nova configuração acessando a url http://localhost:<PORTA_APP>/.

- 13** Adicione uma rota no projeto **CamadaDeControle**. Para isso, modifique o arquivo **RouteConfig.cs** da pasta **App_Start**.

```

1 ...
2 routes.MapRoute(
3     name: "ProdutoListaPorPreco",
4     url: "Produto/Lista/{PrecoMinimo}/{PrecoMaximo}",
5     defaults: new { controller = "Produto", action = "Lista" }
6 );
7 ...

```

Código C# 6.30: RouteConfig.cs

OBS: A rota **ProdutoListaPorPreco** deve ser adicionada antes da rota **Default**.

- 14** Altere a ação **Lista** do controlador **Produto**.

```

1 ...
2 public ActionResult Lista(double? precoMinimo, double? precoMaximo)
3 {
4     K19Context ctx = new K19Context();
5     var produtos = ctx.Produtos.AsEnumerable();
6
7     if (precoMinimo != null && precoMaximo != null)
8     {
9         produtos = from p in produtos
10            where p.Preco >= precoMinimo & p.Preco <= precoMaximo
11            select p;
12     }
13
14     return View(produtos);
15 }
16 ...

```

Código C# 6.31: ProdutoController.cs

Teste a aplicação acessando a url http://localhost:<PORTA_APP>/Produto/Lista/0/200. Altere os valores 0 e 200 para obter listas de produtos diferentes.

- 15** Adicione duas ações no controlador **Produto** para implementar a edição de produtos.

```

1 ...
2 public ActionResult Editar(int id = 0)
3 {
4     K19Context ctx = new K19Context();
5     Produto p = ctx.Produtos.Find(id);

```

```

6     if (p == null)
7     {
8         return HttpNotFound();
9     }
10
11    return View(p);
12 }
13
14 [HttpPost]
15 public ActionResult Editar(Produto p)
16 {
17     K19Context ctx = new K19Context();
18     ctx.Entry(p).State = System.Data.EntityState.Modified;
19
20     ctx.SaveChanges();
21
22     return RedirectToAction("Lista");
23 }
24 ...
25

```

Código C# 6.32: *ProdutoController.cs*

- 16** Modifique a página de listagem de produtos. Para isso, altere o arquivo **Lista.cshtml** da pasta **Views/Produto**.

```

1 @using CamadaDeControle.Models
2 @model IEnumerable<Produto>
3
4 @{
5     ViewBag.Title = "Lista";
6 }
7
8 <h2>Lista</h2>
9
10 @if ( TempData["Mensagem"] != null )
11 {
12     <div>@TempData["Mensagem"]</div>
13 }
14
15 <table>
16     <tr>
17         <th>@Html.DisplayNameFor(m => m.Nome)</th>
18         <th>@Html.DisplayNameFor(m => m.Descricao)</th>
19         <th>@Html.DisplayNameFor(m => m.Preco)</th>
20         <th></th>
21     </tr>
22     @foreach(var produto in Model)
23     {
24         <tr>
25             <td>@Html.DisplayFor(m => produto.Nome)</td>
26             <td>@Html.DisplayFor(m => produto.Descricao)</td>
27             <td>@Html.DisplayFor(m => produto.Preco)</td>
28             <td>@Html.ActionLink("Editar", "Editar", new { id = produto.ProdutoID })</td>
29         </tr>
30     }
31 </table>

```

Código C# 6.33: *Lista.cs*

- 17** Adicione uma página associada à ação **Editar** do controlador **Produto**.

```

1 @model CamadaDeControle.Models.Produto
2

```

```
3  @{
4      ViewBag.Title = "Editar";
5  }
6
7 <h2>Editar</h2>
8
9 @using (Html.BeginForm())
{
10
11     @Html.EditorFor(m => m.ProdutoID)
12     <div>@Html.LabelFor(m => m.Nome)</div>
13     @Html.EditorFor(m => m.Nome)
14     <div>@Html.LabelFor(m => m.Descricao)</div>
15     @Html.EditorFor(m => m.Descricao)
16     <div>@Html.LabelFor(m => m.Preco)</div>
17     @Html.EditorFor(m => m.Preco)
18
19     <div><input type="submit" value="Editar" /></div>
20 }
```

Código C# 6.34: Editar.cs

Teste a aplicação acessando a url http://localhost:<PORTA_APP>/. Altere os dados de alguns produtos.

VALIDAÇÃO

Os usuários podem cometer erros ao preencher um formulário. Por exemplo, esquecer de preencher um campo que é obrigatório. Os parâmetros enviados pelos usuários devem ser validados pela aplicação com o intuito de não permitir o armazenamento de informações erradas.

Controller

O primeiro passo para implementar a validação dos parâmetros enviados através de formulários HTML é definir a lógica de validação na camada de controle.

```
1 if (editora.Nome == null || editora.Nome.Trim().Length == 0)
2 {
3     // Erro de Validação
4 }
```

Código C# 7.1: Definindo as regras de validação

O segundo passo é definir mensagens informativas para enviar aos usuários. O ASP.NET MVC possui um objeto especializado no armazenamento de mensagens de erros de validação. Esse objeto pode ser acessado através da propriedade ModelState.

```
1 if (editora.Nome == null || editora.Nome.Trim().Length == 0)
2 {
3     ModelState.AddModelError("Nome", "O campo Nome é obrigatório");
4 }
```

Código C# 7.2: Definindo as mensagens de erro de validação

As mensagens são armazenadas no ModelState através do método AddModelError(). Esse método permite que as mensagens sejam agrupadas logicamente, pois ele possui dois parâmetros: o primeiro é o grupo da mensagem e o segundo é a mensagem propriamente.

O código de validação pode ser colocado nos controladores, mais especificamente nas ações. Se algum erro for encontrado, o fluxo de execução pode ser redirecionado para uma página que mostre as mensagens informativas aos usuários. Normalmente, essa página é a mesma do formulário que foi preenchido incorretamente. O ModelState possui uma propriedade que indica se erros foram adicionados ou não. Essa propriedade chama-se IsValid.

```
1 [HttpPost]
2 public ActionResult Salva(Editora editora)
3 {
4     if (editora.Nome == null || editora.Nome.Trim().Length == 0)
5     {
6         ModelState.AddModelError("Nome", "O campo Nome é obrigatório.");
7     }
8     if (ModelState.IsValid)
9     {
```

```

10     db.Editoras.Add(editora);
11     TempData["mensagem"] = "A editora foi cadastrada com sucesso!";
12     return RedirectToAction("Index");
13 }
14 else
15 {
16     return View("Cadastra");
17 }
18 }
```

Código C# 7.3: EditoraController.cs

O ASP.NET MVC também pode adicionar mensagens no ModelState antes do controlador ser chamado. Normalmente, essas mensagens estão relacionadas a erros de conversão. Por exemplo, um campo que espera um número é preenchido com letras.

View

As mensagens de erros de validação podem ser acrescentadas em uma página através do método ValidationSummary() da propriedade Html. É importante salientar que esse método adiciona todas as mensagens de erro.

```

1 @model K19.Models.Editora
2 @{
3     ViewBag.Title = "Cadastro de Editora";
4 }
5
6 <h2>Cadastro de Editora</h2>
7
8 @using (Html.BeginForm()) {
9     @Html.ValidationSummary()
10
11     <br/>
12
13     @Html.LabelFor(model => model.Nome)
14     @Html.EditorFor(model => model.Nome)
15
16     <br/>
17
18     @Html.LabelFor(model => model.Email)
19     @Html.EditorFor(model => model.Email)
20
21     <br/>
22
23     <input type="submit" value="Enviar" />
24 }
```

Código CSHTML 7.1: Cadastra.cshtml

Podemos utilizar o método ValidationMessageFor() para mostrar somente as mensagens de erro de validação de um determinado grupo.

Para não mostrar erros dos grupos com o ValidationSummary(), devemos passar como parâmetro o valor true. Nesse caso, apenas os erros que não estão associados a um grupo específico serão apresentados pelo ValidationSummary().

```

1 @model LivrariaVirtual.Models.Editora
2 @{
3 }
```

```

4   ViewBag.Title = "Cadastro de Editora";
5 }
6
7 <h2>Cadastro de Editora</h2>
8
9 @using (Html.BeginForm()) {
10   @Html.ValidationSummary(true)
11
12   <br />
13
14   @Html.LabelFor(model => model.Nome)
15   @Html.EditorFor(model => model.Nome)
16   @Html.ValidationMessageFor(model => model.Nome)
17
18   <br />
19
20   @Html.LabelFor(model => model.Email)
21   @Html.EditorFor(model => model.Email)
22   @Html.ValidationMessageFor(model => model.Email)
23
24   <br />
25
26   <input type="submit" value="Enviar" />
27 }
```

Código CSHTML 7.2: Cadastra.cshtml



Exercícios de Fixação

- 1 Crie um projeto do tipo **ASP.NET MVC 4 Web Application** chamado **Validacao** no **Visual Studio 2012 Express for Web**. Utilize o template **Basic Project**.
- 2 Adicione uma entidade chamada **Jogador** na pasta **Models** do projeto **Validacao**.

```

1 namespace Validacao.Models
2 {
3   public class Jogador
4   {
5     [HiddenInput(DisplayValue=false)]
6     public int JogadorID { get; set; }
7
8     public string Nome { get; set; }
9
10    public int? Numero { get; set; }
11
12    public double? Altura { get; set; }
13  }
14 }
```

Código C# 7.4: Jogador.cs

- 3 Adicione um **DbContext** para registrar a entidade **Jogador** e utilizar os recursos do Entity Framework. Crie uma classe chamada **K19Context** na pasta **Models**.

```

1 namespace Validacao.Models
2 {
3   public class K19Context : DbContext
4   {
```

```

5     public DbSet<Jogador> Jogadores { get; set; }
6 }
7 }
```

Código C# 7.5: K19Context.cs

- 4 Adicione um controlador chamado **Jogador** utilizando o template **Empty MVC Controller**.

```

1 namespace Validacao.Controllers
2 {
3     public class JogadorController : Controller
4     {
5         public ActionResult Lista()
6         {
7             K19Context ctx = new K19Context();
8             return View(ctx.Jogadores);
9         }
10
11        [HttpGet]
12        public ActionResult Cadastra()
13        {
14            return View();
15        }
16
17        [HttpPost]
18        public ActionResult Cadastra(Jogador j)
19        {
20            K19Context ctx = new K19Context();
21            ctx.Jogadores.Add(j);
22            ctx.SaveChanges();
23
24            return RedirectToAction("Lista");
25        }
26    }
27 }
```

Código C# 7.6: JogadorController.cs

- 5 Adicione uma tela associada à ação **Lista** do controlador **Jogador** com o seguinte conteúdo.

```

1 @using Validacao.Models
2 @model IEnumerable<Jogador>
3
4 @{
5     ViewBag.Title = "Lista";
6 }
7
8 <h2>Lista</h2>
9
10 <table>
11     <tr>
12         <th>@Html.DisplayNameFor(m => m.Nome)</th>
13         <th>@Html.DisplayNameFor(m => m.Numero)</th>
14         <th>@Html.DisplayNameFor(m => m.Altura)</th>
15     </tr>
16     @foreach(var jogador in Model)
17     {
18         <tr>
19             <td>@Html.DisplayFor(m => jogador.Nome)</td>
20             <td>@Html.DisplayFor(m => jogador.Numero)</td>
21             <td>@Html.DisplayFor(m => jogador.Altura)</td>
22         </tr>
23     }
24 </table>
```

Código HTML 7.1: Lista.cshtml

- 6 Adicione uma tela associada à ação **Cadastra** do controlador **Jogador** com o seguinte conteúdo.

```

1 @model Validacao.Models.Jogador
2
3 @{
4     ViewBag.Title = "Cadastra";
5 }
6
7 <h2>Cadastra</h2>
8
9 @using (Html.BeginForm())
10 {
11     <div>@Html.LabelFor(m => m.Nome)</div>
12     @Html.EditorFor(m => m.Nome)
13     <div>@Html.LabelFor(m => m.Numero)</div>
14     @Html.EditorFor(m => m.Numero)
15     <div>@Html.LabelFor(m => m.Altura)</div>
16     @Html.EditorFor(m => m.Altura)
17
18     <div><input type="submit" value="Cadastrar" /></div>
19 }
```

Código HTML 7.2: Cadastra.cshtml

Para ver o resultado, acesse a url http://localhost:<PORTA_APP>/Jogador/Cadastra.

- 7 Altere a ação **Cadastra** do controlador **Jogador**. Aplique algumas regras de validação.

```

1 ...
2 [HttpPost]
3 public ActionResult Cadastra(Jogador j)
4 {
5     if (String.IsNullOrEmpty(j.Nome))
6     {
7         ModelState.AddModelError("Nome",
8             "O nome do jogador é obrigatório");
9     }
10    if (j.Numero == null || j.Numero <= 0 || j.Numero >= 100)
11    {
12        ModelState.AddModelError("Numero",
13            "O número do jogador deve ser maior que 0 e menor que 100");
14    }
15    if (j.Altura == null || j.Altura < 0)
16    {
17        ModelState.AddModelError("Altura",
18            "A altura do jogador não pode ser negativa");
19    }
20
21    if (ModelState.IsValid)
22    {
23        K19Context ctx = new K19Context();
24        ctx.Jogadores.Add(j);
25        ctx.SaveChanges();
26        return RedirectToAction("Lista");
27    }
28    else
29    {
30        return View("Cadastra", j);
31    }
32 }
```

33 ...

Código C# 7.7: JogadorController.cs

- 8** Altere o formulário de cadastro de jogador. Para isso, modifique o arquivo **Cadastra.cshtml** da pasta **Views/Jogador**.

```

1 @model Validacao.Models.Jogador
2
3 @{
4     ViewBag.Title = "Cadastra";
5 }
6
7 <h2>Cadastra</h2>
8
9 @using (Html.BeginForm())
{
10     @Html.ValidationSummary(true)
11
12     <div>@Html.LabelFor(m => m.Nome)</div>
13     @Html.EditorFor(m => m.Nome)
14     @Html.ValidationMessageFor(m => m.Nome)
15
16     <div>@Html.LabelFor(m => m.Numero)</div>
17     @Html.EditorFor(m => m.Numero)
18     @Html.ValidationMessageFor(m => m.Numero)
19
20     <div>@Html.LabelFor(m => m.Altura)</div>
21     @Html.EditorFor(m => m.Altura)
22     @Html.ValidationMessageFor(m => m.Altura)
23
24     <div><input type="submit" value="Cadastrar" /></div>
25 }
```

Código CSHTML 7.3: Cadastra.cshtml

Teste as validações cadastrando alguns jogadores através da url http://localhost:<PORTA_APP>/Jogador/Cadastra.

Anotações

As lógicas de validação também podem ser definidas através de anotações adicionadas nas classes de modelo. Dessa forma, essas lógicas não estariam mais nos controladores, o que conceitualmente é o ideal, pois nos controladores só deveria existir lógica para controlar o fluxo da execução.

Required

Uma das validações mais comuns é a de campo obrigatório. Ela pode ser realizada através da anotação **Required** para propriedades do tipo string.

```

1 public class Editora
2 {
3     [Required]
4     public string Nome { get; set; }
5
6     ...
7 }
```

Código C# 7.8: Editora.cs

Com essa anotação, a lógica de validação pode ser retirada do controlador Editora.

```

1 ...
2 [HttpPost]
3 public ActionResult Salva(Editora editora)
4 {
5     if (ModelState.IsValid)
6     {
7         db.Editoras.Add(editora);
8         return RedirectToAction("Index");
9     }
10    else
11    {
12        return View("Cadastra", editora);
13    }
14 }
15 ...

```

Código C# 7.9: EditoraController.cs

Alterando a mensagem

As anotações de validação possuem mensagens padrão que podem ser alteradas através do atributo `ErrorMessage`.

```

1 ...
2 [Required(ErrorMessage="O campo Nome é obrigatório")]
3 public string Nome { get; set; }
4 ...

```

Outros validadores

Há outras anotações para validação:

- Range
- RegularExpression
- StringLength

Validação no lado do Cliente

As validações podem ser realizadas também nos navegadores para melhorar a interação com os usuários.

Antes da terceira versão do ASP.NET MVC, era necessário habilitar a validação no lado do cliente através do método `Html.EnableClientValidation()`. A partir da terceira versão do ASP.NET MVC, a validação no cliente está habilitada por padrão.

Para que a validação no lado cliente funcione corretamente, devemos acrescentar as bibliotecas javascript necessárias. No ASP.NET MVC 4, basta acrescentar a seção de scripts nas páginas.

```

1 @section Scripts {
2     @Scripts.Render("~/bundles/jqueryval")
3 }

```



Exercícios de Fixação

- 9 Altere a entidade **Jogador** acrescentando as anotações de validação.

```

1 namespace Validacao.Models
2 {
3     public class Jogador
4     {
5         [HiddenInput(DisplayValue = false)]
6         public int JogadorID { get; set; }
7
8         [Required(ErrorMessage="O nome do jogador é obrigatório")]
9         public string Nome { get; set; }
10
11        [Required(ErrorMessage="O número do jogador é obrigatório")]
12        [Range(1, 99,
13            ErrorMessage="O número do jogador deve ser maior que 0 e menor que 100")]
14        public int? Numero { get; set; }
15
16        [Required(ErrorMessage="A altura do jogador é obrigatória")]
17        [Range(0, double.MaxValue,
18            ErrorMessage="A altura do jogador não pode ser negativa")]
19        public double? Altura { get; set; }
20    }
21 }
```

Código C# 7.11: *Jogador.cs*

- 10 Altere a ação **Cadastra** do controlador **Jogador**.

```

1 ...
2 [HttpPost]
3 public ActionResult Cadastra(Jogador j)
4 {
5     if (ModelState.IsValid)
6     {
7         K19Context ctx = new K19Context();
8         ctx.Jogadores.Add(j);
9         ctx.SaveChanges();
10        return RedirectToAction("Lista");
11    }
12    else
13    {
14        return View("Cadastra", j);
15    }
16 }
17 ...
```

Código C# 7.12: *JogadorController.cs*

Importante: Apague a base de dados **Validacao.Models.K19Context**

Para ver o resultado, acesse a url http://localhost:<PORTA_APP>/Jogador/Cadastra.

- 11 Para adicionar a validação no lado do cliente, basta alterar a página de cadastro de jogadores. Observe o código abaixo.

```
1 @model Validacao.Models.Jogador
2
3 @{
4     ViewBag.Title = "Cadastra";
5 }
6
7 <h2>Cadastra</h2>
8
9 @using (Html.BeginForm())
{
10     @Html.ValidationSummary(true)
11
12     <div>@Html.LabelFor(m => m.Nome)</div>
13     @Html.EditorFor(m => m.Nome)
14     @Html.ValidationMessageFor(m => m.Nome)
15
16     <div>@Html.LabelFor(m => m.Numero)</div>
17     @Html.EditorFor(m => m.Numero)
18     @Html.ValidationMessageFor(m => m.Numero)
19
20     <div>@Html.LabelFor(m => m.Altura)</div>
21     @Html.EditorFor(m => m.Altura)
22     @Html.ValidationMessageFor(m => m.Altura)
23
24     <div><input type="submit" value="Cadastrar" /></div>
25
26     @section Scripts {
27         @Scripts.Render("~/bundles/jqueryval")
28     }
29 }
```

Código CSHTML 7.5: Cadastra.cshtml

Para testar, acesse a url http://localhost:<PORTA_APP>/Jogador/Cadastra.



SESSÃO

Considere a aplicação de uma loja virtual. Nessa aplicação, os clientes selecionam os produtos desejados e os adiciona no seu carrinho de compra. Cada cliente deve ter o seu próprio carrinho para que os seus produtos não se misturem com os produtos selecionados por outros clientes. A aplicação deve armazenar o carrinho de um cliente até que a compra seja finalizada ou até ela ter certeza que o cliente não precisa mais do carrinho.

Para resolver esse problema, podemos utilizar o conceito de **Sessão**. Para cada navegador conectado, o servidor manterá uma sessão aberta. Dessa forma, podemos separar os dados de cada usuário conectado.

Identificando os navegadores

Para aplicar a ideia de Sessão, é necessário ter a capacidade de identificar o navegador que está requisitando a aplicação a cada requisição. Uma primeira abordagem seria utilizar o endereço IP da máquinas para identificar os navegadores. Porém, nessa abordagem, dois navegadores executando na mesma máquina não poderiam ser identificados individualmente.

Outra abordagem é deixar a cargo do servidor a criação de um identificador único para cada navegador conectado. Quando um navegador faz a primeira requisição para a aplicação, o servidor deve gerar um identificador único para esse navegador e enviá-lo na resposta HTTP. A partir da segunda requisição, os navegadores devem enviar para a aplicação o identificador recebido na primeira requisição. Desta maneira, a aplicação saberá qual é o navegador que está realizando uma requisição. Os navegadores podem enviar os seus respectivos identificadores de diferentes formas. As mais utilizadas são:

Reescrita de URL Nesta abordagem, os identificadores são embutidos nos links e botões das páginas da aplicação. Quando os links ou botões são clicados pelo usuário, o identificador é enviado para a aplicação. Uma desvantagem é que todas as páginas devem ser geradas dinamicamente para adicionar o identificador em todos os links e botões.

Cookies Cookies são arquivos contendo informações. Eles são gerados nos servidores e enviados para os navegadores. Os navegadores armazenam os cookies localmente na máquina do usuário. Além disso, os navegadores enviam os cookies de volta para o servidor em todas as requisições. Os servidores podem armazenar os identificadores gerados em cookies. Dessa forma, a cada requisição, o servidor receberá um cookie contendo o identificador.

Sessões no ASP.NET MVC

No ASP.NET MVC, o objeto que representa uma sessão é um dicionário. Para armazenar informações, você deve adicionar uma chave e um valor na propriedade Session. Considere um objeto da classe Usuario que agrupa as informações sobre um determinado usuário. O código a seguir é um exemplo de como podemos guardar esse objeto na sessão após a realização da autenticação do usuário.

```
1 public class LoginController : Controller
2 {
3     ...
4     public ActionResult Login(Cliente cliente)
5     {
6         ...
7         Session["Cliente"] = cliente;
8         ...
9     }
10 }
```

Código C# 8.1: LoginController.cs

Você pode adicionar qualquer tipo de valor na sessão. De forma análoga, para resgatar as informações armazenadas, basta acessar a chave correspondente da propriedade Session. Veja o exemplo a seguir:

```
1 Cliente cliente = (Cliente)Session["Cliente"];
2 string saudacao = "Bem vindo " + cliente.Nome;
```

Código C# 8.2: Recuperando informações da sessão

Quando um usuário deslogar, podemos eliminar a informação armazenada em sua sessão. Para isso, podemos simplesmente remover todas as chaves do dicionário como no exemplo a seguir.

```
1 Session.RemoveAll();
```

Código C# 8.3: Eliminando todas as informações da sessão

Contudo, o método RemoveAll() não elimina a sessão. Ele apenas remove os dados contidos na sessão. Para eliminá-la, você deve utilizar o método Abandon(). Observe o exemplo a seguir.

```
1 public class LoginController : Controller
2 {
3     ...
4     public ActionResult Logout()
5     {
6         ...
7         Session.Abandon();
8         ...
9     }
10 }
```

Código C# 8.4: LoginController.cs

Session Mode

O ASP.NET MVC disponibiliza quatro modos de sessão (*Session Modes*):

- InProc

- StateServer
- SQLServer
- Custom

A diferença entre eles é o modo de armazenamento das sessões.

No modo InProc, todas as informações da sessão são armazenadas na memória do servidor. Esse é o modo mais simples e mais utilizado, e vem configurado como o padrão. Uma desvantagem desse modo é a possível sobrecarga de memória se forem armazenadas muitas informações na sessão. Por isso não é indicado para sistemas muito grandes, com muitos usuários navegando simultaneamente. Outro problema é que o servidor da aplicação não pode ser desligado, pois a informação armazenada na memória será perdida.

No modo StateServer, as informações são serializadas e enviadas para um servidor independente do servidor da aplicação. Isto possibilita que o servidor de aplicação possa ser reiniciado sem que as sessões sejam perdidas. Uma desvantagem é o tempo gasto para realizar a serialização e deserialização.

No modo SQLServer, as informações também são serializadas, mas são armazenadas em um banco de dados. Além de permitir que o servidor da aplicação seja reiniciado, este modo possibilita um maior controle de segurança dos dados da sessão. Uma desvantagem importante é que o processo é naturalmente lento.

No modo Custom, todo o processo de identificação de usuários e armazenamento de sessões pode ser personalizado.

Para selecionar algum dos modos disponíveis, basta adicionar a tag `<sessionState>` dentro da tag `<system.web>`, no arquivo `Web.Config` da raiz de um projeto ASP.NET MVC 4.

```
1 <sessionState mode="InProc" />
```

Código XML 8.1: Web.Config

Podemos personalizar nossa sessão, modificando alguns atributos da tag `<sessionState>`. Por exemplo, podemos determinar o tempo de timeout das sessões. Veja o exemplo abaixo.

```
1 <sessionState mode="InProc" timeout="30" />
```

Código XML 8.2: Web.Config

Outra possibilidade é desabilitar o uso de cookie com o atributo `cookieless="true"`. Neste caso será utilizada a reescrita de URL.

```
1 <sessionState mode="InProc" cookieless="true" />
```

Código XML 8.3: Web.Config



Exercícios de Fixação

- 1 Crie um projeto do tipo **ASP.NET MVC 4 Web Application** chamado **Sessao** no **Visual Studio 2012 Express for Web**. Utilize o template **Basic Project**.

- 2 Adicione uma entidade chamada **Produto** na pasta **Models** do projeto **Sessao**.

```
1 namespace Sessao.Models
2 {
3     public class Produto
4     {
5         public int ProdutoID { get; set; }
6         public string Nome { get; set; }
7         public double Preco { get; set; }
8     }
9 }
```

Código C# 8.5: *Produto.cs*

- 3 Adicione uma classe chamada **K19Context** na pasta **Models** para registrar a entidade **Produto** e utilizar os recursos do Entity Framework. Lembre-se de fazer com que sua classe herde da classe **DbContext**.

```
1 namespace Sessao.Models
2 {
3     public class K19Context:DbContext
4     {
5         public DbSet<Produto> Produtos { get; set; }
6     }
7 }
```

Código C# 8.6: *K19Context.cs*

- 4 Adicione uma classe chamada **Carrinho** na pasta **Models** do projeto **Sessao**.

```
1 namespace Sessao.Models
2 {
3     public class Carrinho
4     {
5         public List<Produto> Produtos { get; set; }
6
7         public Carrinho()
8         {
9             this.Produtos = new List<Produto>();
10        }
11    }
12 }
```

Código C# 8.7: *Carrinho.cs*

- 5 Adicione um controlador chamado **Produto** baseado na entidade **Produto** e no **DbContext K19Context** utilizando o recurso do *scaffolding*.

- 6 Altere a tela **Index** do controlador **Produto** de acordo com o código abaixo.

```
1 @model IEnumerable<Sessao.Models.Produto>
2
```

```

3  @{
4      ViewBag.Title = "Index";
5  }
6
7 <h2>Index</h2>
8
9 @if (TempData["Mensagem"] != null)
10 {
11     <div>@ TempData["Mensagem"]</div>
12 }
13
14 <p>
15     @Html.ActionLink("Create New", "Create")
16 </p>
17 <table>
18     <tr>
19         <th>
20             @Html.DisplayNameFor(model => model.Nome)
21         </th>
22         <th>
23             @Html.DisplayNameFor(model => model.Preco)
24         </th>
25         <th></th>
26     </tr>
27
28@foreach (var item in Model) {
29     <tr>
30         <td>
31             @Html.DisplayFor(modelItem => item.Nome)
32         </td>
33         <td>
34             @Html.DisplayFor(modelItem => item.Preco)
35         </td>
36         <td>
37             @Html.ActionLink("Edit", "Edit", new { id=item.ProdutoID }) |
38             @Html.ActionLink("Details", "Details", new { id=item.ProdutoID }) |
39             @Html.ActionLink("Delete", "Delete", new { id=item.ProdutoID }) |
40             @Html.ActionLink("Adicionar ao carrinho", "Adicionar", "Carrinho", new { id=item.ProdutoID }, null)
41         </td>
42     </tr>
43 }
44 </table>

```

Código CSHTML 8.1: Index.cshtml

- 7 Adicione um controlador chamado **Carrinho** utilizando o template **Empty MVC Controller**. Em seguida altere-o de acordo com o código abaixo.

```

1 namespace Sessao.Controllers
2 {
3     public class CarrinhoController : Controller
4     {
5         //
6         // GET: /Carrinho/
7
8         public ActionResult Index()
9         {
10             return View(this.PegaCarrinhoDaSessao());
11         }
12
13         public ActionResult Cancelar()
14         {
15             Session.Abandon();
16             return RedirectToAction("Index", "Produto");
17         }
18

```

```

19     public ActionResult Adicionar(int id = 0)
20     {
21         K19Context ctx = new K19Context();
22         Produto p = ctx.Produtos.Find(id);
23
24         if (p == null)
25         {
26             return HttpNotFound();
27         }
28
29         Carrinho carrinho = this.PegaCarrinhoDaSessao();
30         carrinho.Produtos.Add(p);
31
32         TempData["Mensagem"] = "Produto adicionado ao carrinho com sucesso!";
33
34         return RedirectToAction("Index", "Produto");
35     }
36
37     public ActionResult Remover(int idx = 0)
38     {
39         Carrinho carrinho = this.PegaCarrinhoDaSessao();
40         carrinho.Produtos.RemoveAt(idx);
41
42         return RedirectToAction("Index");
43     }
44
45     private Carrinho PegaCarrinhoDaSessao()
46     {
47         if (Session["Carrinho"] == null)
48         {
49             Session["Carrinho"] = new Carrinho();
50         }
51
52         return Session["Carrinho"] as Carrinho;
53     }
54 }

```

Código C# 8.8: CarrinhoController.cs

- 8 Adicione uma tela associada à ação **Index** do controlador **Carrinho** com o seguinte conteúdo.

```

1 @model Sessao.Models.Carrinho
2 @{
3     ViewBag.Title = "Index";
4 }
5 <h2>Index</h2>
6
7 @Html.ActionLink("Cancelar carrinho", "Cancelar")
8
9 <ul>
10    @{
11        int i = 0;
12        foreach (var produto in Model.Produtos)
13        {
14            <li>@produto.Nome (@produto.Preco) - @Html.ActionLink("Remover do carrinho",
15            "Remover", new { idx = i++ })</li>
16        }
17    }
18 </ul>

```

Código CSHTML 8.2: Index.cshtml

- 9 Altere o layout **_Layout** de acordo com o código abaixo.

```
1 @using Sessao.Models;
2 <!DOCTYPE html>
3 <html>
4 <head>
5     <meta charset="utf-8" />
6     <meta name="viewport" content="width=device-width" />
7     <title>@ViewBag.Title</title>
8     @Styles.Render("~/Content/css")
9     @Scripts.Render("~/bundles/modernizr")
10    </head>
11    <body>
12        <div>
13            @{
14                Carrinho carrinho = Session["Carrinho"] as Carrinho;
15
16                if(carrinho != null){
17                    @Html.ActionLink("Meu carrinho (" + @carrinho.Produtos.Count + " produtos)",
18                        "Index", "Carrinho")
19                }
20            }
21        </div>
22        @RenderBody()
23
24        @Scripts.Render("~/bundles/jquery")
25        @RenderSection("scripts", required: false)
26    </body>
27    </html>
```

Código CSHTML 8.3: _Layout.cshtml



AUTENTICAÇÃO

Muitas vezes, queremos restringir o acesso à determinadas áreas de uma aplicação, seja por questões de segurança ou por organização. Em nossa aplicação poderíamos, por exemplo, definir que para poder adicionar, alterar e remover editoras, o usuário deve estar logado no sistema. Caso contrário, o usuário apenas poderá listar as editoras.

```
1 ...
2 public ActionResult Cadastra()
3 {
4     if (Session["Cliente"] != null)
5     {
6         return base.View();
7     }
8     else
9     {
10        return base.RedirectToAction("Index", "Login");
11    }
12 }
13 ...
```

Código C# 9.1: EditoraController.cs

No exemplo acima, caso um usuário tente adicionar uma editora através do formulário de cadastro, a ação `Cadastra` verificará se o usuário já está logado. Essa verificação é realizada através do uso da sessão, como visto no capítulo 8. Se o usuário não estiver logado, ele será redirecionado para a página de Login. Apesar de funcionar, este código apresenta uma inconveniência. Temos que adicionar essa lógica em todas as ações que queremos que tenha o mesmo comportamento, ou seja, que apenas permitam o acesso de usuários logados.

Em outros casos, podemos desejar que algumas ações executem alguma tarefa em comum. Por exemplo, poderíamos adicionar uma mensagem em um arquivo de logging sempre que uma ação fosse executada. Desse modo, poderíamos guardar um histórico sobre o que a aplicação mais executou, qual foi a página mais visitada, etc. Mas novamente, teríamos que adicionar a mesma tarefa em todas as ações da nossa aplicação.

Nesses casos, em que várias ações possuem um comportamento em comum, podemos utilizar o conceito de filtros. Um filtro é semelhante a um método que é executado antes ou depois que uma ação.

Filtro de Autenticação

O ASP.NET MVC já possui alguns filtros prontos para serem utilizados, como o filtro de autenticação. Podemos utilizar ele para o nosso primeiro exemplo, onde exigimos que o usuário esteja logado (autenticado) para acessar determinadas áreas da aplicação. Para isso, precisamos adicionar o seguinte código no nosso método de login:

```

1 ...
2 FormsAuthentication.SetAuthCookie(usuario.Username, false);
3 ...

```

Código C# 9.2: AutenticadorController.cs

Isto adiciona um novo cookie utilizado para a autenticação do usuário. Este novo cookie é independente do cookie utilizado para armazenar informações da sessão. O primeiro parâmetro é referente ao nome do usuário (ou algo que o identifique). O segundo parâmetro é um booleano relativo ao tipo do cookie, se é permanente ou não. Caso seja true, ele sempre irá considerar que o usuário está autenticado após a primeira autenticação.

Para eliminar o cookie de autenticação, devemos adicionar o seguinte código na ação de logout:

```

1 ...
2 FormsAuthentication.SignOut();
3 ...

```

Código C# 9.3: AutenticadorController.cs

As ações que devem utilizar o filtro de autenticação devem ser anotadas com Authorize.

```

1 ...
2 [Authorize]
3 public ActionResult Cadastra()
4 {
5     return base.View();
6 }
7 ...

```

Código C# 9.4: EditoraController.cs

Se queremos aplicar o mesmo filtro a todas as ações de um controlador, podemos adicionar a notação Authorize na classe.

```

1 [Authorize]
2 public class EditoraController : Controller
3 {
4     ...
5 }

```

Código C# 9.5: EditoraController.cs

Desse modo, todas as ações presentes no controlador da Editora exigem que o usuário esteja autenticado.

Quando o filtro de autenticação “barra” um usuário de acessar uma página, podemos redirecioná-lo para a página de login. Devemos incluir o seguinte código dentro da tag <system.web> do arquivo Web.Config.

```

1 <authentication mode="Forms">
2     <forms loginUrl="~/Autenticador/Formulario" timeout="2880"/>
3 </authentication>

```

Código XML 9.1: Web.Config

Para verificar se o usuário associado à sessão atual está autenticada, podemos utilizar a propriedade `IsAuthenticated`, como a seguir:

```
1 if (User.Identity.IsAuthenticated)
2 {
3     ...
4 }
```

Código C# 9.6: Verificando se o usuário está autenticado

Podemos pegar a informação de quem está autenticado através do seguinte comando:

```
1 string nome = User.Identity.Name;
```

Código C# 9.7: Verificando quem é o usuário autenticado

Isto irá pegar o nome que passamos como parâmetro para o método `SetAuthCookie` na autenticação.



Exercícios de Fixação

- 1 Crie um projeto do tipo **ASP.NET MVC 4 Web Application** chamado **Autenticacao** no **Visual Studio 2012 Express for Web**. Utilize o template **Basic Project**.
- 2 Adicione uma entidade chamada **Produto** na pasta **Models** do projeto **Autenticacao**.

```
1 namespace Autenticacao.Models
2 {
3     public class Produto
4     {
5         public int ProdutoID { get; set; }
6         public string Nome { get; set; }
7         public double Preco { get; set; }
8     }
9 }
```

Código C# 9.8: Produto.cs

- 3 Adicione uma classe chamada **K19Context** na pasta **Models** para registrar a entidade **Produto** e utilizar os recursos do Entity Framework. Lembre-se de fazer com que sua classe herde da classe `DbContext`.

```
1 namespace Autenticacao.Models
2 {
3     public class K19Context : DbContext
4     {
5         public DbSet<Produto> Produtos { get; set; }
6     }
7 }
```

Código C# 9.9: K19Context.cs

4 Adicione uma classe **Usuario** na pasta **Models**.

```

1  using System.ComponentModel.DataAnnotations;
2
3  namespace Autenticacao.Models
4  {
5      public class Usuario
6      {
7          public string Username { get; set; }
8          [DataType(DataType.Password)]
9          public string Password { get; set; }
10     }
11 }
```

Código C# 9.10: *Usuario.cs*

5 Adicione o controlador **Autenticador** para permitir a autenticação do usuário.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using System.Web.Security;
7  using Autenticacao.Models;
8
9  namespace Autenticacao.Controllers
10 {
11     public class AutenticadorController : Controller
12     {
13
14         public ActionResult Formulario()
15         {
16             return View();
17         }
18
19         public ActionResult Entrar(Usuario usuario)
20         {
21             if (usuario.Username != null && usuario.Password != null &&
22                 usuario.Username.Equals("K19") && usuario.Password.Equals("K19"))
23             {
24
25                 Session["Usuario"] = usuario;
26                 return RedirectToAction("Index", "Produto");
27             }
28             else
29             {
30                 ViewBag.Mensagem = "usuário ou senha incorretos";
31                 return View("Formulario");
32             }
33         }
34
35         public ActionResult Sair()
36         {
37             Session.Abandon();
38             return RedirectToAction("Formulario");
39         }
40     }
41 }
```

Código C# 9.11: *AutenticadorController.cs*

```

1  @model Autenticacao.Models.Usuario
2
3  @{
4      ViewBag.Title = "Formulario";
```

```

5  }
6
7 <h2>Login</h2>
8
9 @using (Html.BeginForm("Entrar", "Autenticador"))
10 {
11     @Html.ValidationSummary(true)
12
13     <fieldset>
14         <legend>Usuario</legend>
15
16         <div class="editor-label">
17             @Html.LabelFor(model => model.Username)
18         </div>
19         <div class="editor-field">
20             @Html.EditorFor(model => model.Username)
21             @Html.ValidationMessageFor(model => model.Username)
22         </div>
23
24         <div class="editor-label">
25             @Html.LabelFor(model => model.Password)
26         </div>
27         <div class="editor-field">
28             @Html.EditorFor(model => model.Password)
29             @Html.ValidationMessageFor(model => model.Password)
30         </div>
31
32         <p>
33             <input type="submit" value="Logar" />
34         </p>
35     </fieldset>
36 }
37
38 <div>
39     @Html.ActionLink("Back to List", "Index")
40 </div>
41
42 @section Scripts {
43     @Scripts.Render("~/bundles/jqueryval")
44 }

```

Código CSHTML 9.1: Formulario.cshtml

- 6 Adicione um controlador para cadastrar um produto e listar os produtos. Altere o controlador para permitir que apenas os usuários logados possam cadastrar os produtos.

```

1 namespace Autenticacao.Controllers
2 {
3     public class ProdutoController : Controller
4     {
5         private K19Context db = new K19Context();
6
7         public ActionResult Index()
8         {
9             return View(db.Produtos.ToList());
10        }
11
12        public ActionResult Cadastrar()
13        {
14            if (Session["Usuario"] != null)
15            {
16                return View();
17            }
18            else
19            {
20                TempData["Mensagem"] = "Acesso não permitido";
21                return RedirectToAction("Index");
22            }
23        }
24    }
25 }

```

```

22     }
23
24     }
25
26     [HttpPost]
27     public ActionResult Cadastrar(Produto produto)
28     {
29         if (ModelState.IsValid)
30         {
31             db.Produtos.Add(produto);
32             db.SaveChanges();
33             return RedirectToAction("Index");
34         }
35         return View(produto);
36     }
37
38
39 }
40

```

Código C# 9.12: *ProdutoController.cs*

```

1 @model IEnumerable<Autenticacao.Models.Produto>
2 @{
3     ViewBag.Title = "Index";
4 }
5
6
7 <h2>Index</h2>
8 @if (TempData["Mensagem"] != null)
9 {
10     <h3>@ TempData["Mensagem"]</h3>
11 }
12 <h3></h3>
13 <p>
14     @Html.ActionLink("Cadastrar Produto", "Cadastrar")
15 </p>
16 <table>
17     <tr>
18         <th>
19             @Html.DisplayNameFor(model => model.Nome)
20         </th>
21         <th>
22             @Html.DisplayNameFor(model => model.Preco)
23         </th>
24         <th></th>
25     </tr>
26
27     @foreach (var item in Model)
28     {
29         <tr>
30             <td>
31                 @Html.DisplayFor(modelItem => item.Nome)
32             </td>
33             <td>
34                 @Html.DisplayFor(modelItem => item.Preco)
35             </td>
36
37         </tr>
38     }
39
40 </table>

```

Código CSHTML 9.2: *Index.cshtml*

```

1 @model Autenticacao.Models.Produto
2 @{
3

```

```

4     ViewBag.Title = "Cadastrar";
5 }
6
7 <h2>Cadastrar</h2>
8
9 @using (Html.BeginForm())
{
10     @Html.ValidationSummary(true)
11
12     <fieldset>
13         <legend>Produto</legend>
14
15         <div class="editor-label">
16             @Html.LabelFor(model => model.Nome)
17         </div>
18         <div class="editor-field">
19             @Html.EditorFor(model => model.Nome)
20             @Html.ValidationMessageFor(model => model.Nome)
21         </div>
22
23         <div class="editor-label">
24             @Html.LabelFor(model => model.Preco)
25         </div>
26         <div class="editor-field">
27             @Html.EditorFor(model => model.Preco)
28             @Html.ValidationMessageFor(model => model.Preco)
29         </div>
30
31         <p>
32             <input type="submit" value="Create" />
33         </p>
34     </fieldset>
35 }
36
37 <div>
38     @Html.ActionLink("Back to List", "Index")
39 </div>
40
41 @section Scripts {
42     @Scripts.Render("~/bundles/jqueryval")
43 }

```

Código CSHTML 9.3: Cadastrar.cshtml

Para testar, acesse a url http://localhost:<PORTA_APP>/Produto/ e clique no link de cadastrar. Depois acesse o endereço http://localhost:<PORTA_APP>/Autenticador/Formulario/ para fazer a autenticação através do usuário **K19** e senha **K19**.

- 7 Ao invés de usar sessão, vamos utilizar o filtro de autenticação pronto do ASP.NET MVC. Primeiro, devemos alterar as ações Entrar e Sair do controlador Autenticador.

```

1 using System.Web.Mvc;
2 using System.Web.Security;
3 using K19.Models;
4
5 namespace K19.Controllers
6 {
7     public class AutenticadorController : Controller
8     {
9
10         public ActionResult Formulario()
11         {
12             return View();
13         }
14

```

```

15     public ActionResult Entrar(Usuario usuario)
16     {
17         if (usuario.Username != null && usuario.Password != null &&
18             usuario.Username.Equals("K19") && usuario.Password.Equals("K19")) {
19
20             FormsAuthentication.SetAuthCookie(usuario.Username, false);
21             return RedirectToAction("Index", "Produto");
22         }
23         else
24         {
25             ViewBag.Mensagem = "usuário ou senha incorretos";
26             return View("Formulario");
27         }
28     }
29
30     public ActionResult Sair()
31     {
32         FormsAuthentication.SignOut();
33         return RedirectToAction("Formulario");
34     }
35 }
36

```

Código C# 9.13: AutenticadorController.cs

- 8 Altere o controlador Produto para que somente usuários logados possam acessar o formulário de cadastro.

```

1 ...
2 [Authorize]
3 public ActionResult Cadastra()
4 {
5     return View();
6 }
7 ...

```

Código C# 9.14: ProdutoController.cs

- 9 Defina a url para a qual os usuários não logados serão redirecionados ao tentarem acessar o formulário de cadastro de editoras. Altere o arquivo Web.config da raiz do projeto **Autenticacao**.

```

1 ...
2 <authentication mode="Forms">
3     <forms loginUrl="~/Autenticador/Formulario" timeout="2880" />
4 </authentication>
5 ...

```

Código XML 9.2: Web.config

Acesse o endereço http://localhost:<PORTA_APP>/Produto/Cadastra. Verifique o funcionamento do filtro de autenticação que redirecionará os usuários que não estão logados para a url definida no arquivo Web.config.

TRATAMENTO DE ERROS

Inevitavelmente, as aplicações estão sujeitas a erros de várias naturezas. Por exemplo, erros podem ser gerados pelo preenchimento incorreto dos campos de um formulário. Esse tipo de erro é causado por falhas dos usuários. Nesse caso, é importante mostrar mensagens informativas com o intuito de fazer o próprio usuário corrigir os valores preenchidos incorretamente. Veja o Capítulo 7.

Por outro lado, há erros que não são causados por falhas dos usuários. Por exemplo, um erro de conexão com o banco de dados. Nesses casos, é improvável que os usuários possam fazer algo que resolva o problema. E mesmo que pudesse, provavelmente, não seria conveniente esperar que eles o fizessem.

Quando um erro desse tipo ocorre, o ASP.NET MVC cria uma página web com informações sobre o erro e a envia aos usuários. Para usuários locais, o ASP.NET MVC envia uma página web com informações detalhadas do erro ocorrido. Para usuários remotos, a página web enviada não contém informações detalhadas.

Em geral, não é conveniente que os usuários recebam detalhes técnicos sobre os erros gerados por falhas da aplicação. A primeira justificativa é que esses detalhes podem confundir os usuários.

Server Error in '/' Application.

Login failed for user 'sa'.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information.

Exception Details: System.Data.SqlClient.SqlException: Login failed for user 'sa'.

Source Error:

```
Line 13:         public void Adiciona(Editora e)
Line 14:             {
Line 15:                 context.Editoras.Add(e);
Line 16:                 context.SaveChanges();
Line 17:             }
```

Source File: C:\Users\Marcelo\Documents\Visual Studio 2010\Projects\LivrariaVirtual\LivrariaVirtual\Models\EditoraRepository.cs **Line:** 15

Stack Trace:

```
[SqlException (0x80131904): Login failed for user 'sa'.]
System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection)
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning() +234
System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataHandler, SqlConnection connection, Boolean enlistOK) +35
System.Data.SqlClient.SqlInternalConnectionTds.CompleteLogin(Boolean enlistOK) +52
System.Data.SqlClient.SqlInternalConnectionTds.AttemptOneLogin(ServerInfo serverInfo, Boolean failIfLogonFailure, Boolean ignoreFailIfLogonFailure, Boolean validateUserName, Boolean validateUserPassword, Boolean& mustCloseConnection) +200
System.Data.SqlClient.SqlInternalConnectionTds.LoginNoFailover(ServerInfo serverInfo, Boolean failIfLogonFailure, Boolean ignoreFailIfLogonFailure, Boolean validateUserName, Boolean validateUserPassword) +112
System.Data.SqlClient.SqlInternalConnectionTds.OpenLoginEnlist(SqlConnection owningObject, SqlConnectionString connectionOptions, String newPassword, Boolean validate, Boolean& mustCloseConnection) +102
System.Data.SqlClient.SqlInternalConnectionTds..ctor(DbConnectionPoolIdentity identity, SqlConnectionString connectionOptions, Object providerInfo, String newPassword, SqlConnectionStringBuilder options, Boolean validate, Boolean encrypt, Boolean trustServerCertificate, Boolean integratedSecurity) +152
System.Data.SqlClient.SqlConnectionFactory.CreateConnection(DbConnectionOptions options, DbConnectionPool pool, Object providerInfo, String newPassword, SqlConnectionStringBuilder options) +52
System.Data.ProviderBase.DbConnectionFactory.CreatePooledConnection(DbConnection owningObject) +52
System.Data.ProviderBase.DbConnectionFactory.CreateObject(DbConnection owningObject) +42
System.Data.ProviderBase.DbConnectionPool.GetConnection(DbConnection owningObject) +42
System.Data.ProviderBase.DbConnectionFactory.GetConnection(DbConnection owningConnection) +42
```

Figura 10.1: Exemplo de uma página web com informações sobre um erro

A segunda justificativa é que esses detalhes podem expor alguma falha de segurança da aplicação, deixando-a mais vulnerável a ataques.

Try-Catch

Os erros de aplicação podem ser identificados através do comando try-catch, que pode ser colocados nos métodos que definem as ações dos controladores. Ao identificar a ocorrência de um erro, os controladores podem devolver uma página web com alguma mensagem para o usuário.

```

1 ...
2 [HttpPost]
3 public ActionResult Salva(Editora editora)
4 {
5     try
6     {
7         db.Editoras.Add(editora);
8     }
9     catch
10    {
11        return View("Error");
12    }
13
14    return RedirectToAction("Index");
15 }
16 ...

```

Código C# 10.1: EditoraController.cs

Podemos criar uma página `Error.cshtml` na pasta **Views\Shared**. Dessa forma, todo controlador poderá devolver essa página.

```

1 @{
2     Layout = null;
3 }
4
5 <!DOCTYPE html>
6 <html>
7 <head>
8     <title>Erro</title>
9 </head>
10 <body>
11     <h2>
12         Servidor com problemas
13     </h2>
14     <p>
15         Houve um problema em nosso servidor.<br/>
16         Por favor tente novamente dentro de alguns instantes.
17     </p>
18 </body>
19 </html>

```

Código CSHTML 10.1: Error.cshtml

As páginas de erro que serão mostradas pelos controladores teriam uma mensagem simples informando que houve um erro na aplicação e que não é possível atender a requisição do usuário naquele momento. Inclusive, seria conveniente padronizar a página de erro. Em outras palavras, todos os controladores teriam que mostrar a mesma página.

Custom Errors

Utilizar o comando try-catch nos controladores para lidar com os erros de aplicação não é uma boa alternativa, pois o código do controlador fica mais complexo. Além disso, haveria replicação de código nos controladores, pois provavelmente a página de erro seria padronizada.

Para lidar com os erros de aplicação de uma maneira mais prática e fácil de manter, podemos configurar o ASP.NET MVC para utilizar páginas de erro padrão. O primeiro passo é alterar o arquivo de configuração Web.config, acrescentando a tag <customErrors> dentro da tag <system.web>.

```

1 ...
2 <customErrors mode="On">
3 </customErrors>
4 ...
5 ...

```

Código XML 10.1: Web.Config

O atributo mode da tag <customErrors> pode assumir três valores:

On: A página de erro padrão será enviada para usuários locais e remotos.

Off: A página de erro detalhada será enviada para usuários locais e remotos.

RemoteOnly: A página de erro detalhada será enviada para os usuários locais e a padrão para os remotos.

Por convenção, o ASP.NET MVC mantém uma página de erro padrão dentro da pasta **Views\Shared** com o nome **Error.cshtml**. Vamos alterar este arquivo. O conteúdo da página de erro é basicamente HTML.

```

1 @{
2     Layout = null;
3 }
4
5 <!DOCTYPE html>
6 <html>
7 <head>
8     <title>Erro</title>
9 </head>
10 <body>
11     <h2>
12         Servidor com problemas
13     </h2>
14     <p>
15         Houve um problema no nosso servidor.<br />
16         Por favor tente novamente dentro de alguns instantes.
17     </p>
18 </body>
19 </html>

```

Código CSHTML 10.2: Error.cshtml

Http Errors

Um dos erros mais conhecidos do HTTP é o 404, que ocorre quando o navegador faz uma requisição para uma url que não existe. Basicamente, esse erro é gerado por falhas dos usuários ao tentarem digitar diretamente uma url na barra de endereço dos navegadores ou por links ou botões “quebrados” nas páginas da aplicação.

Quando o erro 404 ocorre, o ASP.NET MVC utiliza a página padrão para erros de aplicação configurada no Web.config através da tag <customErrors>. Porém, esse erro não deve ser considerado um erro de aplicação, pois ele pode ser gerado por falhas do usuário. Ele também não deve ser considerado um erro de usuário, pois ele pode ser gerado por falhas da aplicação. Consequentemente, é comum tratar o erro 404 de maneira particular, criando uma página de erro específica para ele.

```

1 @{
2     Layout = null;
3 }
4
5 <!DOCTYPE html>
6 <html>
7 <head>
8     <title>Arquivo não encontrado!</title>
9 </head>
10 <body>
11     <h2>
12         Esse arquivo não foi encontrado. Verifique se a url está correta.
13     </h2>
14 </body>
15 </html>
```

Código CSHTML 10.3: NotFound.cshtml

No arquivo de configuração, podemos especificar uma página web particular para o erro 404 ou para os outros erros do HTTP.

```

1 ...
2 <customErrors mode="On">
3     <error statusCode="404" redirect="~/ErrorPage/NotFound"/>
4 </customErrors>
5 ...
```

Código XML 10.2: Web.Config

De acordo com o código anterior, devemos definir um controlador chamado ErrorPage com uma ação chamada NotFound. Essa ação será acionada toda vez que o erro 404 do HTTP ocorrer.

```

1 namespace K19.Controllers
2 {
3     public class ErrorPageController : Controller
4     {
5         public ActionResult NotFound()
6         {
7             return View();
8         }
9     }
10 }
```

Código C# 10.2: ErrorPageController.cs



Exercícios de Fixação

- 1** Crie um projeto do tipo **ASP.NET MVC 4 Web Application** chamado **TratamentoDeErros** no **Visual Studio 2012 Express for Web**. Utilize o template **Basic Project**.

- 2** Crie um controlador Teste no projeto **TratamentoDeErros**, conforme o código abaixo.

```

1 namespace TratamentoDeErros.Controllers
2 {
3     public class TesteController : Controller
4     {
5         public ActionResult TestaErro()
6         {
7             string[] nomes = new string[] { "Jonas Hirata", "Rafael Cosentino" };
8             string nome = nomes[2];
9             return View();
10        }
11    }
12 }
```

Código C# 10.3: *TesteController.cs*

Acesse o seguinte endereço http://localhost:<PORTA_APP>/Teste/TestaErro e verifique a tela de erro.

Observação: No Visual Web Developer, quando executamos a aplicação, ele a executa em modo *debug*. Dessa forma, toda vez que um erro for gerado no processamento de uma requisição, a execução da aplicação é suspensa no ponto em que o erro ocorreu e detalhes sobre o problema são apresentados. Para continuar a execução da aplicação após a ocorrência de um erro, aperte a tecla F5.

- 3** Trate o erro do exercício anterior com o bloco *try-catch* e redirecione o usuário para uma tela com a seguinte mensagem “Sistema Temporariamente Indisponível”.

Altere a ação *TestaErro* do controlador *Teste*.

```

1 ...
2 public ActionResult TestaErro()
3 {
4     try
5     {
6         string[] nomes = new string[] { "Jonas Hirata", "Rafael Cosentino" };
7         string nome = nomes[2];
8         return View();
9     }
10    catch
11    {
12        return View("Error");
13    }
14 }
```

Código C# 10.4: *TesteController.cs*

- 4** Altere o código do arquivo *Error.cshtml* da pasta *Views\Shared* conforme o código abaixo:

```

1 @{
2     Layout = null;
3 }
```

```

4  <!DOCTYPE html>
5  <html>
6  <head>
7      <meta name="viewport" content="width=device-width" />
8      <title>Error</title>
9  </head>
10 <body>
11     <h2>
12         Sistema Temporariamente Indisponível...
13     </h2>
14 </body>
15 </html>

```

Código CSHTML 10.4: Error.cshtml

Acesse novamente o endereço http://localhost:<PORTA_APP>/Teste/TestaErro e verifique a nova página de erro.

- 5** Remova o bloco try-catch da ação TestaErro do controlador Teste que você adicionou no exercício anterior.

```

1 ...
2 public ActionResult TestaErro()
3 {
4     string[] nomes = new string[] { "Jonas Hirata", "Rafael Cosentino" };
5     string nome = nomes[2];
6     return View();
7 }
8 ...

```

Código C# 10.5: TesteController.cs

- 6** Altere o arquivo Web.config para configurar o ASP.NET MVC para utilizar páginas de erro padrão.

```

1 ...
2 <system.web>
3     <customErrors mode="On">
4     </customErrors>
5 ...
6 <system.web>
7 ...

```

Código XML 10.3: Web.config

Acesse o endereço http://localhost:<PORTA_APP>/Teste/TestaErro e verifique a página de erro.

- 7** Vamos definir uma página para ser exibida quando o erro 404 ocorrer. Primeiramente, crie um controlador chamado ErrorPage com uma ação chamada NotFound.

```

1 namespace TratamentoDeErros.Controllers
2 {
3     public class ErrorPageController : Controller
4     {
5         public ActionResult NotFound()
6         {

```

```

7     return View();
8 }
9 }
10 }
11 }
```

Código C# 10.6: *ErroPageController.cs*

- 8** Na pasta Views\ErrorPage do projeto **Erros**, adicione um arquivo chamado **NotFound.cshtml** com o seguinte conteúdo.

```

1 @{
2     Layout = null;
3 }
4
5 <!DOCTYPE html>
6 <html>
7 <head>
8     <title>Arquivo não encontrado!</title>
9 </head>
10 <body>
11     <h2>
12         Esse arquivo não foi encontrado. Verifique se a url está correta.
13     </h2>
14 </body>
15 </html>
```

Código CSHTML 10.5: *NotFound.cshtml*

- 9** Altere o arquivo **Web.Config** para definir que quando um erro 404 ocorrer, a página exibida deve ser aquela que acabamos de criar.

```

1 ...
2 <system.web>
3     <customErrors mode="On">
4         <error statusCode="404" redirect="~/ErrorPage/NotFound"/>
5     </customErrors>
6     ...
7 <system.web>
8 ...
```

Código XML 10.4: *Web.config*

Tente acessar páginas da aplicação que não existam. Acesse, por exemplo, o endereço http://localhost:<PORTA_APP>/PaginaInexistente.





ASP.NET WEB API

Para permitir a implementação de webservices restful, o ASP.NET Web API foi adicionado à quarta versão do ASP.NET MVC. Webservices restful são baseados no estilo arquitetural REST. Discutiremos, a seguir, os principais conceitos desse estilo arquitetural.

REST

Resources

No estilo arquitetural REST, toda informação disponível é chamada de **resource**. Por exemplo, uma imagem é um resource. Uma lista de produtos é um resource. O cadastro de uma pessoa é um resource.

URIs

Todo resource possui um identificador único globalmente. Os identificadores são utilizados para acessar os respectivos resources. Particularmente, em uma rede HTTP, os resources são identificados por **URIs**(Uniform Resource Identifier - <http://tools.ietf.org/html/rfc3986>). Por exemplo, a URI www.k19.com.br/cursos identifica na internet a lista de cursos da K19.

Media Types

Um resource pode ser apresentado em diversos formatos. Na arquitetura REST, esses formatos são chamados de **media type**. Considere o cadastro de uma pessoa disponível em uma rede HTTP. Eventualmente, esse cadastro pode ser apresentado em **html**, **xml** ou **json**.

```
1 <html>
2   <head>
3     <title>Rafael Cosentino</title>
4   </head>
5
6   <body>
7     <h1>Rafael Cosentino</h1>
8     <p>Líder de treinamentos da K19</p>
9   </body>
10 </html>
```

```
1 <pessoa>
2   <nome>Rafael Cosentino</nome>
3   <descricao>Líder de treinamentos da K19</descricao>
4 </pessoa>
```

```
1 {"nome": "Rafael Cosentino", "descricao": "Líder de treinamentos da K19"}
```

Operações

Em uma arquitetura REST, um conjunto pequeno e fixo de operações deve ser definido previamente. As operações são utilizadas para manipular os recursos de alguma forma.

Por exemplo, em uma rede HTTP, os recursos são manipulados pelos métodos do protocolo HTTP. Podemos atribuir uma semântica diferente para cada um desses métodos.

Resource	Método HTTP	Semântica
www.k19.com.br/cursos	GET	pega a lista de cursos
www.k19.com.br/cursos	POST	adiciona um curso na lista

Resources

No ASP.NET Web API, os resources são definidos por controladores de uma aplicação ASP.NET MVC que derivam de `System.Web.Http.ApiController`.

```
1 public class CursosController : ApiController
2 {
3     ...
4 }
```

Código C# A.1: CursosController.cs

URIs

As URIs dos resources são definidas através de rotas do ASP.NET Web API criadas no arquivo `RouteConfig.cs`. O Visual Web Developer adiciona a seguinte rota padrão do ASP.NET Web API nos projetos ASP.NET MVC 4:

```
1 routes.MapHttpRoute(
2     name: "DefaultApi",
3     routeTemplate: "api/{controller}/{id}",
4     defaults: new { id = RouteParameter.Optional }
5 );
```

Código C# A.2: RouteConfig.cs

De acordo com a rota padrão do ASP.NET Web API, a URI do resource correspondente ao controlador `Cursos` é `http://localhost:<PORTA_APP>/api/cursos`.

Operações

Podemos associar as operações HTTP aos métodos da classe `CursosController`. Essas associações são estabelecidas automaticamente através dos nomes dos métodos. Por exemplo, a operação

GET do HTTP é associada automaticamente a um método cujo nome possui o prefixo Get. Veja o exemplo abaixo.

```

1 public class CursosController : ApiController
2 {
3     ...
4     // GET http://localhost:<PORTA_APP>/api/cursos
5     public Curso[] GetCursos()
6     {
7         ...
8     }
9 }
```

Código C# A.3: CursosController.cs

Na rota padrão do ASP.NET Web API, um parâmetro opcional chamado id foi definido. Quando esse parâmetro estiver presente em uma requisição do tipo GET, podemos recuperar o valor dele em um método cujo nome possua o prefixo Get e tenha um argumento chamado id. Observe o código abaixo.

```

1 public class CursosController : ApiController
2 {
3     ...
4     // GET http://localhost:<PORTA_APP>/api/cursos/5
5     public Curso[] GetCurso(int id)
6     {
7         ...
8     }
9 }
```

Código C# A.4: CursosController.cs

Também podemos recuperar parâmetros de URL. Para isso, basta definir argumentos com os mesmos nomes desses parâmetros nos métodos associados às operações do HTTP. Veja o exemplo abaixo

```

1 public class CursosController : ApiController
2 {
3     ...
4     // GET http://localhost:<PORTA_APP>/api/cursos?sigla=K32
5     public Curso[] GetCurso(string sigla)
6     {
7         ...
8     }
9 }
```

Código C# A.5: CursosController.cs

Considere o código a seguir.

```

1 public class CursosController : ApiController
2 {
3     ...
4     // POST http://localhost:<PORTA_APP>/api/cursos
5     public void PostCurso(Curso curso)
6     {
7         ...
8     }
9 }
```

10 }

Código C# A.6: CursosController.cs

O método PostCurso() será associado a operação POST do HTTP. Os valores dos parâmetros enviados dentro de uma requisição do tipo POST à url `http://localhost:<PORTA_APP>/api/cursos` serão armazenados automaticamente pelo ASP.NET Web API nas propriedades do argumento curso do método PostCurso() de acordo com o nome dessas propriedades e dos nomes dos parâmetros enviados. Por exemplo, o valor do parâmetro Nome será armazenado dentro da propriedade Nome caso ambos existam.

Media Type

Por padrão, o ASP.NET Web API utiliza os headers Content-type e Accept para definir o media type dos dados de entrada e saída. No exemplo anterior, se uma requisição do tipo GET for realizada à url `//POSThttp://localhost:<PORTA_APP>/api/cursos` com header Accept: application/json, o resource correspondente será apresentado em formato JSON.



Exercícios de Fixação

1 Crie um projeto do tipo **ASP.NET MVC 4 Web Application** chamado **WebApi** no **Visual Studio 2012 Express for Web**. Utilize o template **Basic Project**.

2 Vamos definir o nosso primeiro webservice utilizando web api. Para isso, adicione uma classe para modelar cursos na pasta **Models**.

```
1 namespace WebApi.Models
2 {
3     public class Curso
4     {
5         static int contador;
6
7         public Curso()
8         {
9             Curso.contador++;
10            this.Id = contador;
11        }
12
13        public int Id { get; set; }
14        public string Sigla { get; set; }
15        public string Nome { get; set; }
16    }
17 }
```

Código C# A.7: Curso.cs

3 Agora crie um controlador chamado **Cursos** para implementar as operações do webservice. Na criação desse controlador, selecione o template **Empty API Controller**.

1 using System;

```

2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Net;
5  using System.Net.Http;
6  using System.Web.Http;
7  using LivrariaVirtual.Models;
8  using System.Web.Mvc;
9
10 namespace WebApi.Controllers
11 {
12     public class CursosController : ApiController
13     {
14         private static List<Curso> cursos = new List<Curso>
15         {
16             new Curso
17             {
18                 Sigla = "K31",
19                 Nome = "C# e Orientação a Objetos"
20             },
21             new Curso
22             {
23                 Sigla = "K32",
24                 Nome = "Desenvolvimento Web com ASP .NET MVC"
25             }
26         };
27     }
28
29     public List<Curso> Get()
30     {
31         return cursos;
32     }
33
34     public Curso Get(string sigla)
35     {
36         Curso curso = cursos.Find(c => c.Sigla.Equals(sigla));
37         return curso;
38     }
39
40     public Curso Get(int id)
41     {
42         Curso curso = cursos.Find(c => c.Id == id);
43         return curso;
44     }
45
46     public void Post(Curso curso)
47     {
48         cursos.Add(curso);
49     }
50 }
51 }
```

Código C# A.8: CursosController.cs

- 4** Para testar o webservice que retorna a lista de cursos, basta acessar o seguinte endereço: http://localhost:<PORTA_APP>/api/cursos.
- 5** Teste o webservice que devolve o curso a partir de uma sigla. Basta acessar o seguinte endereço: http://localhost:<PORTA_APP>/api/cursos?sigla=K31.
- 6** Teste o webservice que devolve um curso a partir de um id. Basta acessar o seguinte endereço: http://localhost:<PORTA_APP>/api/cursos/1.

- 7 Defina uma página com um formulário para testar o webservice que adiciona curso. Para isso, crie um controlador Curso conforme o código abaixo:

```
1 using System.Web.Mvc;
2
3 namespace WebApi.Controllers
4 {
5     public class CursoController : Controller
6     {
7         public ActionResult Cadastrar()
8         {
9             return View();
10        }
11    }
12}
13
```

Código C# A.9: *CursoController.cs*

- 8 Para testar o webservice que adiciona curso, devemos definir a página **Cadastrar.cshtml**.

```
1 @model WebApi.Models.Curso
2
3 @{
4     ViewBag.Title = "Cadastrar";
5 }
6
7 <h2>Cadastrar</h2>
8
9 @using (Html.BeginForm(null, null, FormMethod.Post, new { @action = "/api/cursos" }))
10 {
11     @Html.ValidationSummary(true)
12
13     <fieldset>
14         <legend>Curso</legend>
15
16         <div class="editor-label">
17             @Html.LabelFor(model => model.Sigla)
18         </div>
19         <div class="editor-field">
20             @Html.EditorFor(model => model.Sigla)
21             @Html.ValidationMessageFor(model => model.Sigla)
22         </div>
23
24         <div class="editor-label">
25             @Html.LabelFor(model => model.Nome)
26         </div>
27         <div class="editor-field">
28             @Html.EditorFor(model => model.Nome)
29             @Html.ValidationMessageFor(model => model.Nome)
30         </div>
31
32         <p>
33             <input type="submit" value="Create" />
34         </p>
35     </fieldset>
36 }
37
38 <div>
39     @Html.ActionLink("Back to List", "Index")
40 </div>
41
42 @section Scripts {
43     @Scripts.Render("~/bundles/jqueryval")
44 }
```

Código CSHTML A.1: Cadastrar.cshtml

- 9 Teste o webservice que adiciona curso através do formulário criado no exercício anterior. Acesse o formulário através do seguinte endereço: http://localhost:<PORTA_APP>/Curso/Cadastrar. Para verificar se o curso foi adicionado, acesse http://localhost:<PORTA_APP>/api/cursos/.





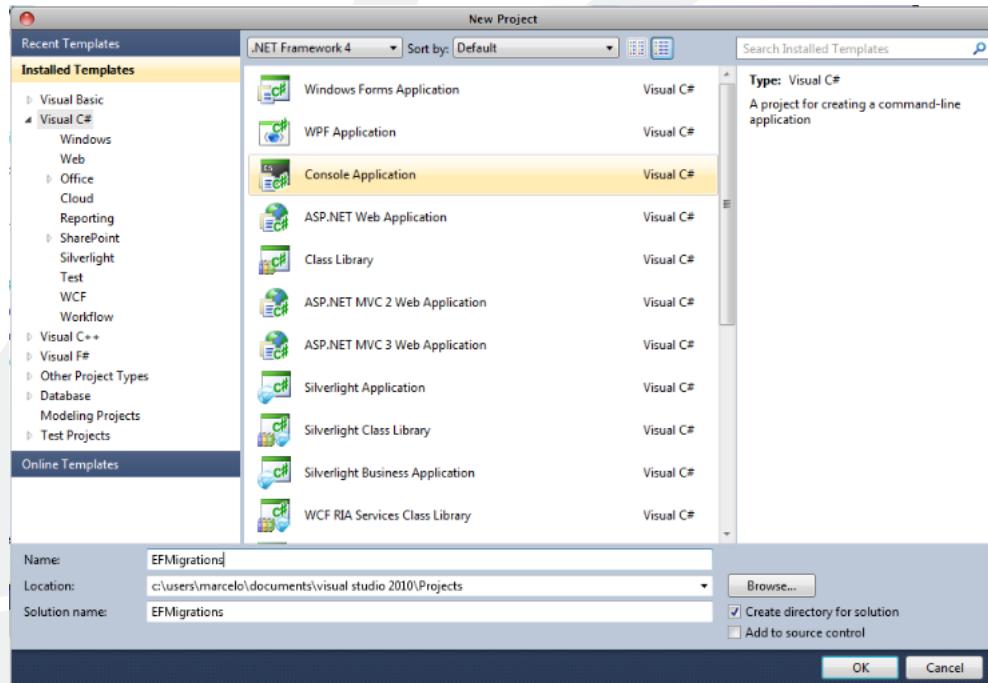
MIGRATIONS

O Entity Framework Code First da Microsoft tem uma funcionalidade que permite controlar as mudanças no banco de dados que serão realizadas de acordo com as alterações na camada de modelo da aplicação. Essa funcionalidade é chamada de **Code First Migrations**.

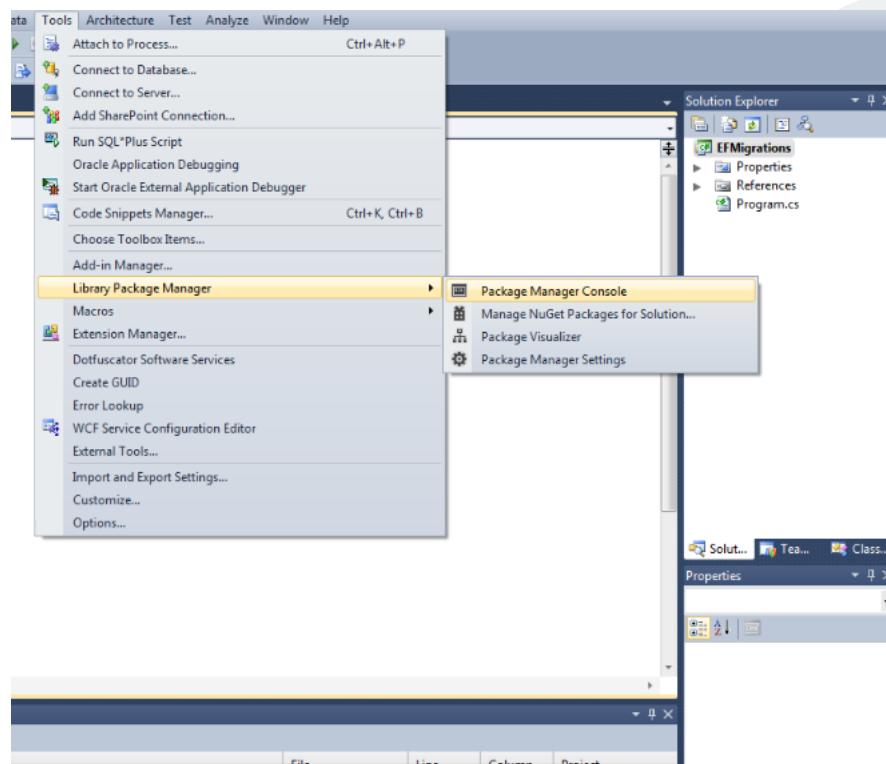
O Code First Migrations foi adicionado na versão 4.3.1 do Entity Framework. Neste capítulo utilizaremos o Entity Framework 5.0.

Passo a Passo

Para testar o funcionamento do Code First Migrations, criaremos um projeto no Visual Studio.



Agora, devemos instalar o Entity Framework 5.0. Para isso, utilizaremos o gerenciador de pacotes Nuget do Visual Studio.



O Nuget permite que bibliotecas e ferramentas sejam instaladas no Visual Studio. Ao instalar pacotes no seu projeto, ele adiciona as bibliotecas, referências e faz as alterações necessárias nos arquivos de configuração. Mais informações sobre o Nuget e os pacotes disponíveis, acesse o endereço <http://nuget.org/packages>

Para instalar o Entity Framework 5.0, basta executar o seguinte comando no Package Manager Console.

```
Install-Package EntityFramework -Pre
```

```
PM> Install-Package EntityFramework -Pre
Você está baixando EntityFramework de Microsoft, para o qual o contrato de licença está disponível em http://go.microsoft.com/fwlink/?LinkId=248059. Verifique o pacote quanto a dependências adicionais, as quais podem vir com seu(s) próprio(s) contrato(s) de licença. O seu uso do pacote e de suas dependências constitui a sua aceitação dos termos dos contratos de licença. Se você não aceitar os termos do(s) contrato(s) de licença, exclua os componentes relevantes do seu dispositivo.
'EntityFramework 5.0.0-rc' instalado com êxito.
Exito ao adicionar 'EntityFramework 5.0.0-rc' a EF.Migrations.

Type 'get-help EntityFramework' to see all available Entity Framework commands.

PM>
```

Após a instalação do Entity Framework 5.0, adicionaremos uma classe de modelo chamada **Editora**. Essa entidade será mapeada através de uma classe chamada **Livraria**. Para testar, criaremos uma classe com o método **Main()**. Observe o código dessas três classes.

```
1 namespace EF.Migrations
2 {
3     public class Editora
4     {
5         public int Id { get; set; }
```

```

6     public string Nome { get; set; }
7     public string Email { get; set; }
8 }
9 }
```

Código C# B.1: Editora.cs

```

1 namespace EFMigrations
2 {
3     public class Livraria : DbContext
4     {
5         public DbSet<Editora> Editoras { get; set; }
6     }
7 }
```

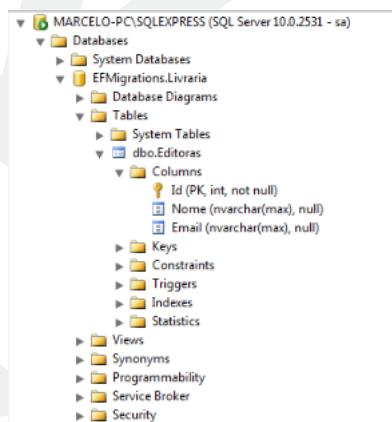
Código C# B.2: Livraria.cs

```

1 namespace EFMigrations
2 {
3     class Program
4     {
5         static void Main(string[] args)
6         {
7             using (Livraria livraria = new Livraria())
8             {
9                 Editora e = new Editora { Nome = "K19", Email = " contato@k19.com.br" };
10                livraria.Editoras.Add(e);
11                livraria.SaveChanges();
12            }
13        }
14    }
15 }
```

Código C# B.3: Program.cs

Após a execução do projeto, teremos a seguinte tabela e banco de dados.



O próximo passo é alterar a classe de modelo Editora.

```

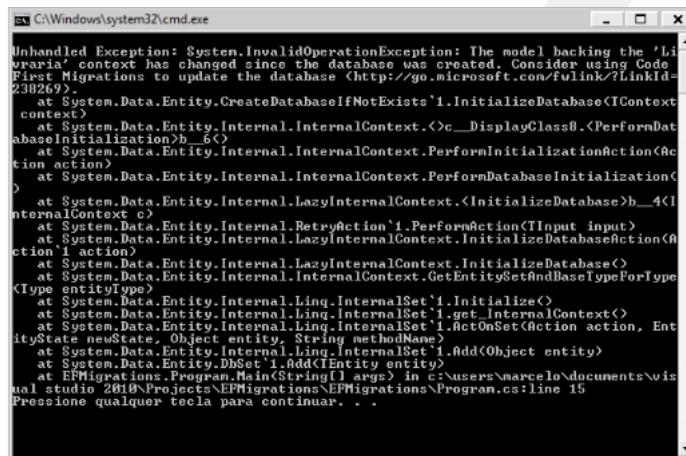
1 namespace EFMigrations
2 {
3     public class Editora
4     {
5         public int Id { get; set; }
6         public string Nome { get; set; }
7         public string Email { get; set; }
8 }
```

```

9     public string Telefone { get; set; }
10    }
11 }
```

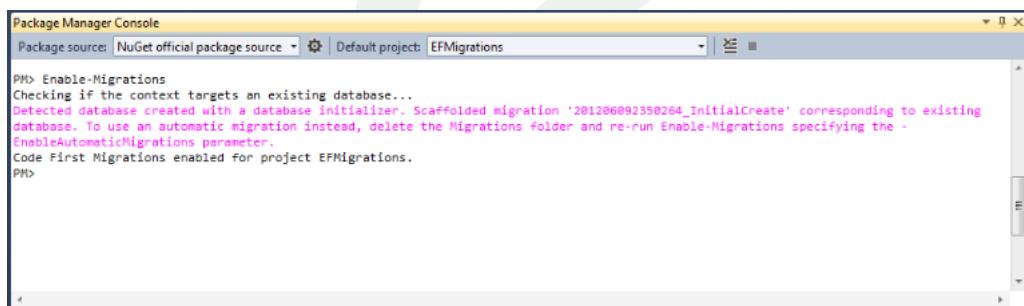
Código C# B.4: Editora.cs

Com a alteração na classe Editora, um exceção é gerada ao executar o projeto.



Para resolver esse problema, o banco de dados precisa ser atualizado. Para fazer essa atualização, o Entity Framework recomenda a utilização da ferramenta Code First Migrations.

O primeiro passo para utilizar Code First Migrations é habilitá-lo e adicioná-lo ao projeto através do Package Manager Console.



O comando visto acima adiciona uma pasta chamada **Migrations** no projeto. Esta pasta contém dois arquivos.

- A classe Configuration permite definir o comportamento do Code First Migrations para o nosso contexto.
- A classe parcial InitialCreate define a primeira versão das tabelas do banco de dados.

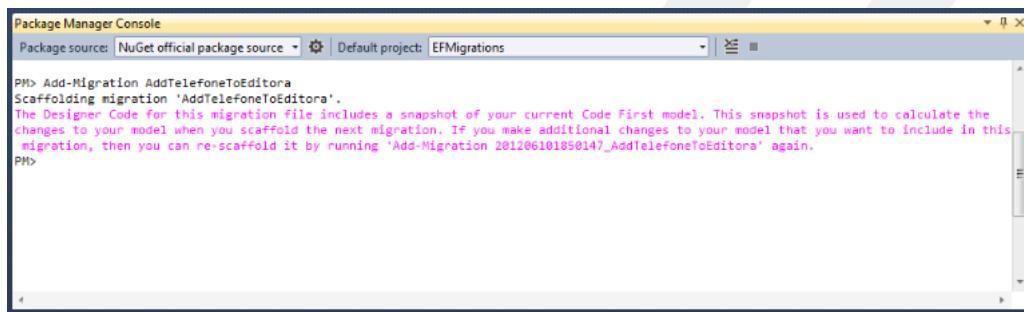
O Code First Migration tem dois comandos fundamentais.

- Add-Migration que gera o código necessário para atualizar o banco de dados de acordo com as alterações nas classes de modelo.

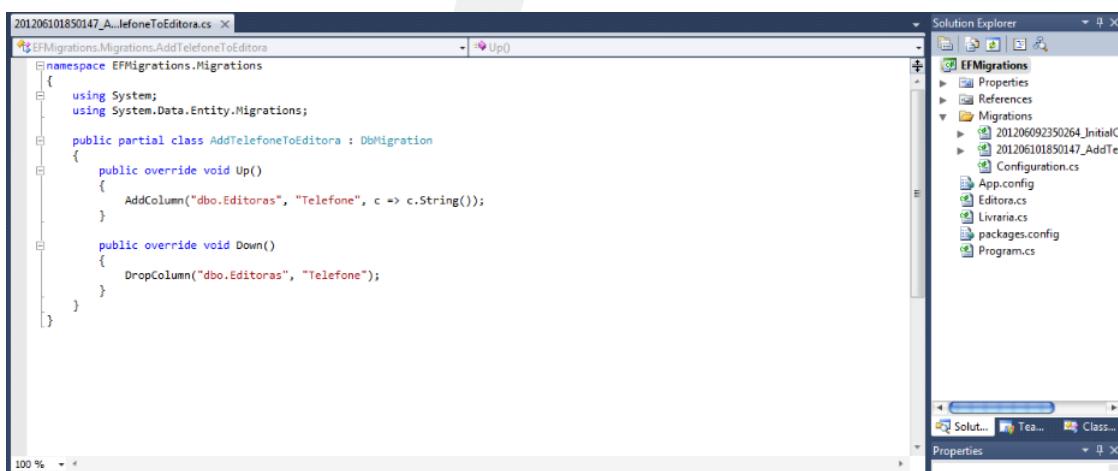
- Update-Database aplica as alterações pendentes no banco de dados.

Como adicionamos a propriedade Telefone na classe de modelo Editora, devemos criar e executar uma migração para atualizar o banco de dados. Para criar uma migração, devemos utilizar o comando Add-Migration. Para executar a migração, devemos utilizar o comando Update-Database.

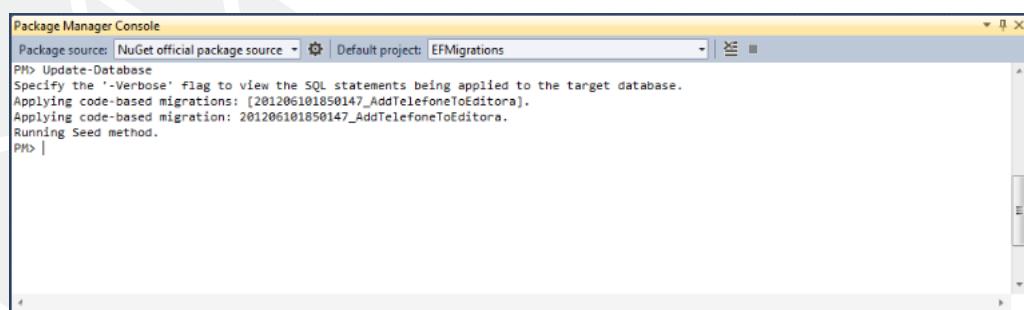
Observe a execução do comando Add-Migration.



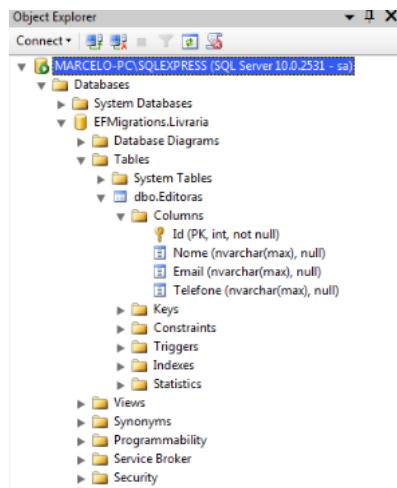
A classe que define a migração é adicionada na pasta **Migrations**



Para aplicar a migração, devemos executar o comando Update-Database.



No banco de dados, uma coluna é adicionada na tabela Editoras.



Podemos também adicionar uma nova entidade. Considere a seguinte classe para definir os livros da nossa aplicação.

```

1 namespace EFMigrations
2 {
3     public class Livro
4     {
5         public int Id { get; set; }
6         public string Titulo { get; set; }
7         public decimal Preco { get; set; }
8         public int EditoraId { get; set; }
9         public Editora Editora { get; set; }
10    }
11 }
```

Código C# B.5: Livro.cs

Para estabelecer o relacionamento entre editoras e livros, a classe `Editora` deve ser alterada.

```

1 namespace EFMigrations
2 {
3     public class Editora
4     {
5         public int Id { get; set; }
6         public string Nome { get; set; }
7         public string Email { get; set; }
8         public string Telefone { get; set; }
9
10        [ForeignKey("EditoraId")]
11        public virtual IList<Livro> Livros { get; set; }
12    }
13 }
```

Código C# B.6: Editora.cs

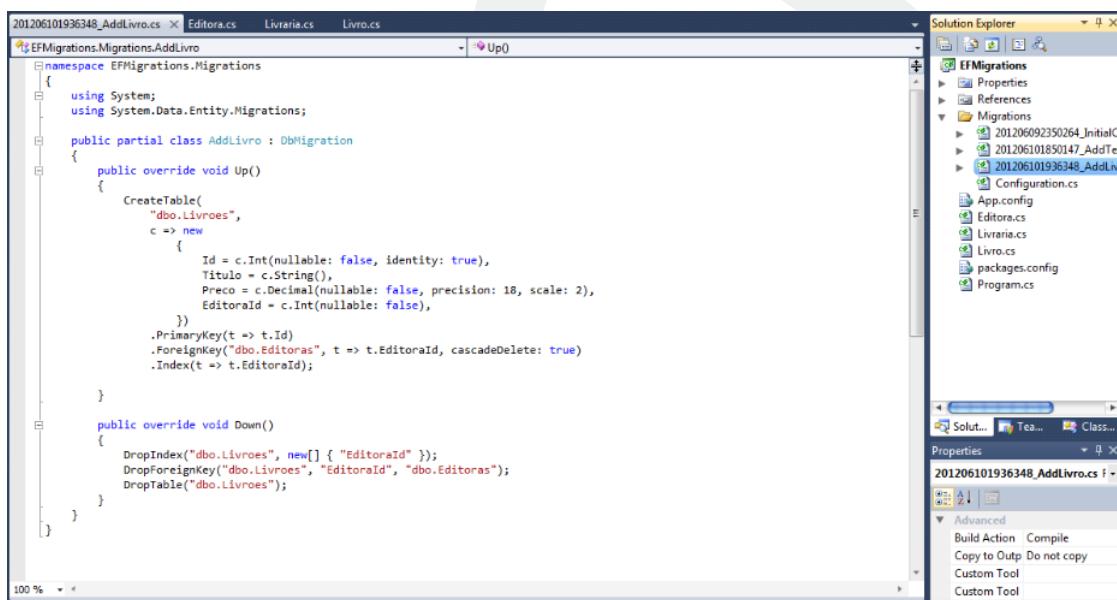
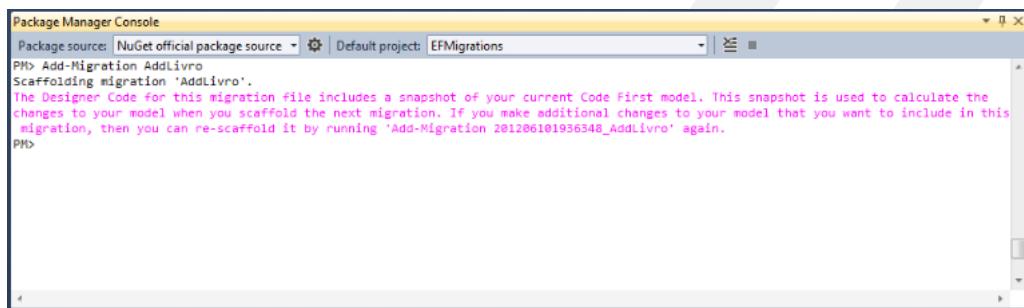
A classe `Livro` deve ser mapeada no contexto `Livraria`.

```

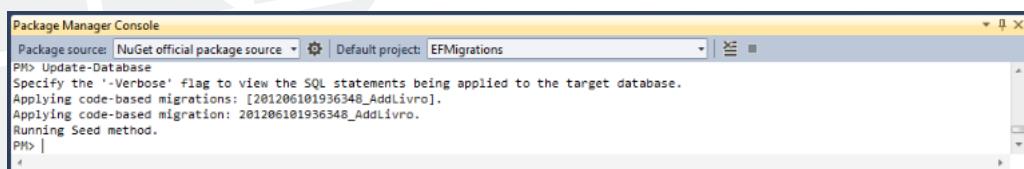
1 namespace EFMigrations
2 {
3     public class Livraria : DbContext
4     {
5         public DbSet<Editora> Editoras { get; set; }
6         public DbSet<Livro> Livros { get; set; }
7     }
8 }
```

Código C# B.7: Livraria.cs

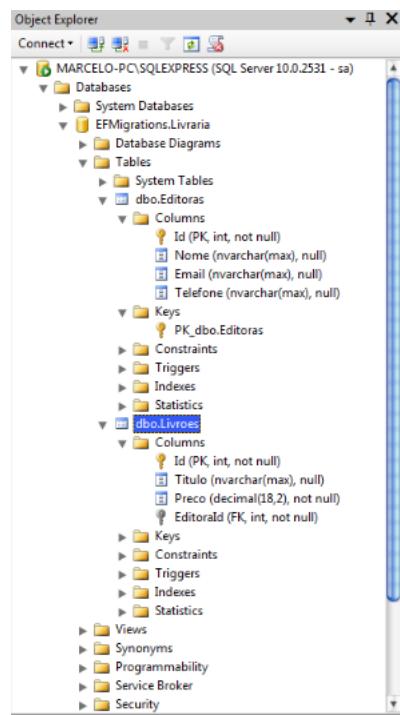
Após a alteração das classes Editora e Livraria e a criação da entidade Livro, devemos criar uma migração para atualizar o banco de dados.



Para aplicar a migração, devemos utilizar o comando `Update-Database`.



A tabela correspondente à classe Livro e a chave estrangeira para realizar o relacionamento entre livros e editoras são adicionadas no banco de dados.



Exercícios de Fixação

1 Crie um projeto do tipo **ASP.NET MVC 4 Web Application** chamado **CodeFirstMigrations** no **Visual Studio 2012 Express for Web**. Utilize o template **Basic Project**.

2 Para testar o Code First Migrations, adicione a classe **Editora** no projeto **CodeFirstMigrations**.

```

1  using System.ComponentModel.DataAnnotations;
2
3 namespace CodeFirstMigrations.Models
4 {
5     public class Editora
6     {
7         public int Id { get; set; }
8         [Required(ErrorMessage = "O campo nome é obrigatório")]
9         public string Nome { get; set; }
10        [Required(ErrorMessage = "O campo email é obrigatório")]
11        public string Email { get; set; }
12    }
13 }
```

Código C# B.8: *Editora.cs*

3 Crie uma classe **K19Context** e adicione a propriedade **DbSet** para mapear a entidade **Editora**.

```

1  using System.Data.Entity;
2
3 namespace CodeFirstMigrations.Models
4 {
```

```
5     public class K19Context : DbContext
6     {
7         public DbSet<Editora> Editoras { get; set; }
8     }
9 }
```

Código C# B.9: K19Context.cs

- 4 Crie um controlador **Editora** para listar e cadastrar as editoras.

```
1 using CodeFirstMigrations.Models;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Web;
6 using System.Web.Mvc;
7
8 namespace CodeFirstMigrations.Controllers
9 {
10    public class EditoraController : Controller
11    {
12        private K19Context db = new K19Context();
13
14        //
15        // GET: /Editora/
16
17        public ActionResult Index()
18        {
19            return View(db.Editoras.ToList());
20        }
21
22
23        //
24        // GET: /Editora/Cadastrar
25
26        public ActionResult Cadastrar()
27        {
28            return View();
29        }
30
31        //
32        // POST: /Editora/Cadastrar
33
34        [HttpPost]
35        public ActionResult Cadastrar(Editora editora)
36        {
37            if (ModelState.IsValid)
38            {
39                db.Editoras.Add(editora);
40                db.SaveChanges();
41                return RedirectToAction("Index");
42            }
43
44            return View(editora);
45        }
46
47
48
49        protected override void Dispose(bool disposing)
50        {
51            db.Dispose();
52            base.Dispose(disposing);
53        }
54
55    }
56}
```

Código C# B.10: EditoraController.cs

- 5 Crie as telas de listagem e cadastro de editoras.

```
1 @model IEnumerable<CodeFirstMigrations.Models.Editora>
2 @{
3     ViewBag.Title = "Index";
4 }
5
6 <h2>Index</h2>
7
8 <p>
9     @Html.ActionLink("Cadastrar Editora", "Cadastrar")
10 </p>
11 <table>
12     <tr>
13         <th>
14             @Html.DisplayNameFor(model => model.Nome)
15         </th>
16         <th>
17             @Html.DisplayNameFor(model => model.Email)
18         </th>
19         <th></th>
20     </tr>
21
22     @foreach (var item in Model)
23     {
24         <tr>
25             <td>
26                 @Html.DisplayFor(modelItem => item.Nome)
27             </td>
28             <td>
29                 @Html.DisplayFor(modelItem => item.Email)
30             </td>
31         </tr>
32     }
33
34 </table>
```

Código CSHTML B.1: Index.cshtml

```
1 @model CodeFirstMigrations.Models.Editora
2 @{
3     ViewBag.Title = "Cadastrar";
4 }
5
6 <h2>Cadastrar</h2>
7
8 @using (Html.BeginForm())
9 {
10     @Html.ValidationSummary(true)
11
12     <fieldset>
13         <legend>Editora</legend>
14
15         <div class="editor-label">
16             @Html.LabelFor(model => model.Nome)
17         </div>
18         <div class="editor-field">
19             @Html.EditorFor(model => model.Nome)
20             @Html.ValidationMessageFor(model => model.Nome)
21         </div>
```

```

23      <div class="editor-label">
24          @Html.LabelFor(model => model.Email)
25      </div>
26      <div class="editor-field">
27          @Html.EditorFor(model => model.Email)
28          @Html.ValidationMessageFor(model => model.Email)
29      </div>
30
31      <p>
32          <input type="submit" value="Cadastrar" />
33      </p>
34  </fieldset>
35 }
36
37 <div>
38     @Html.ActionLink("Back to List", "Index")
39 </div>
40
41 @section Scripts {
42     @Scripts.Render("~/bundles/jqueryval")
43 }
44

```

Código CSHTML B.2: Cadastrar.cshtml

Teste o cadastro de editoras através do endereço http://localhost:<PORTA_APP>/Editora/Cadastrar.

6 Adicione a propriedade **Telefone** à classe Editora.

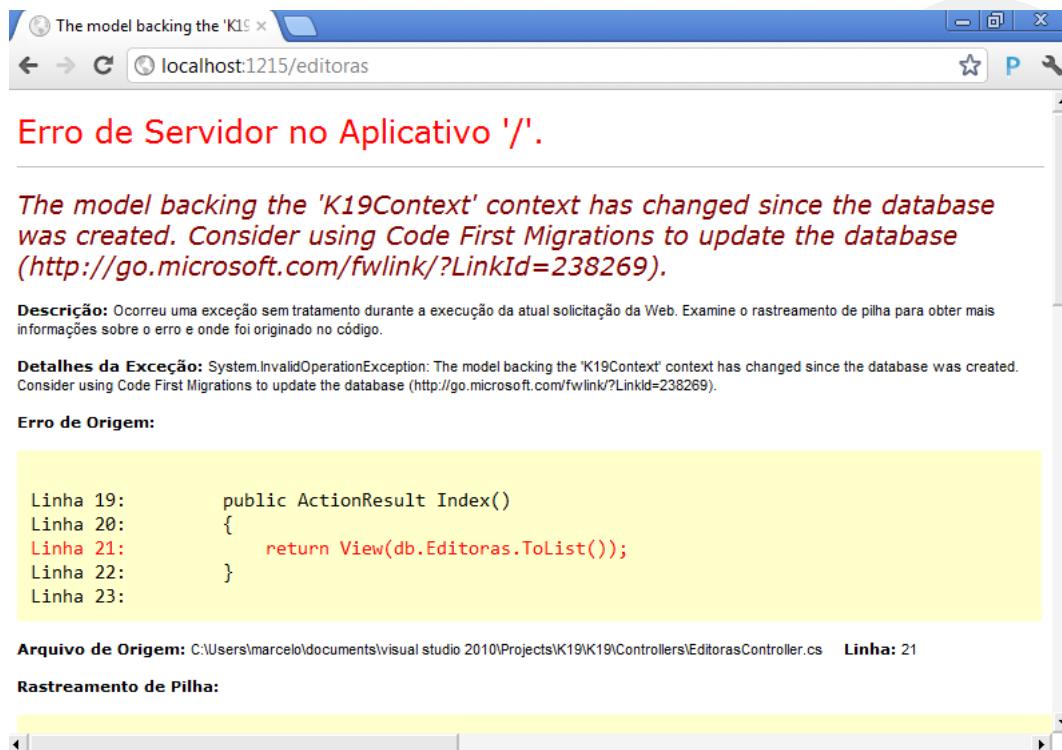
```

1  using System.ComponentModel.DataAnnotations;
2
3 namespace CodeFirstMigrations.Models
4 {
5     public class Editora
6     {
7         public int Id { get; set; }
8         [Required(ErrorMessage = "O campo nome é obrigatório")]
9         public string Nome { get; set; }
10        [Required(ErrorMessage = "O campo email é obrigatório")]
11        public string Email { get; set; }
12        public string Telefone { get; set; }
13    }
14 }

```

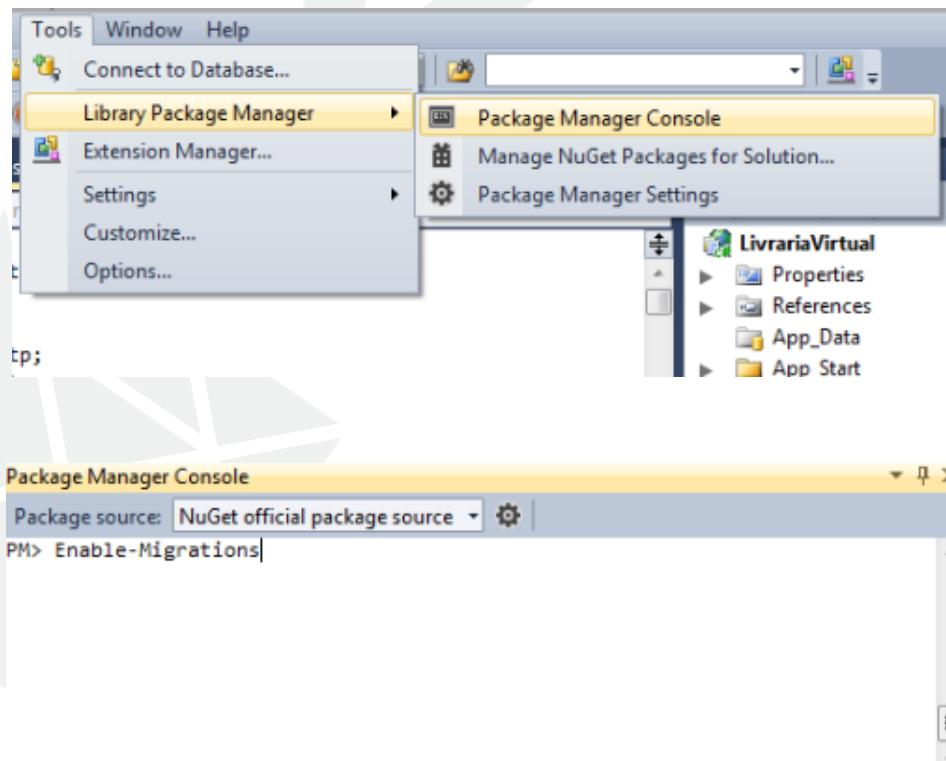
Código C# B.11: Editora.cs

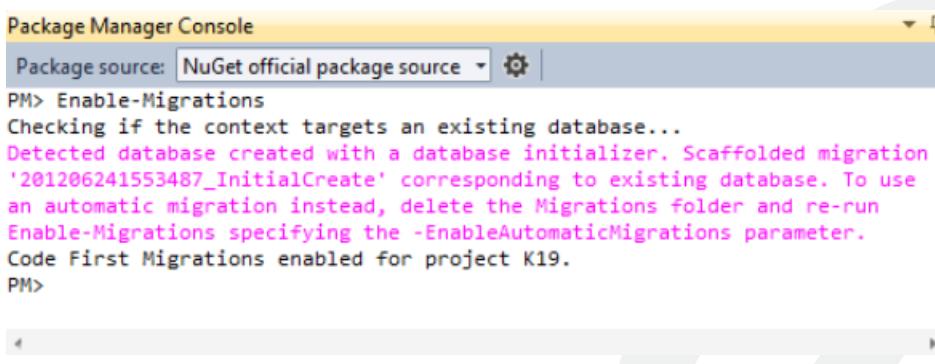
7 Acesse o seguinte endereço: http://localhost:<PORTA_APP>/Editora. Verifique que uma exceção do tipo **System.InvalidOperationException** ocorrerá.



- 8 Para corrigir o problema visto no exercício anterior, devemos habilitar o *Code First Migrations*. Para isso, execute o seguinte comando através do *Package Manager Console*.

Enable-Migrations.

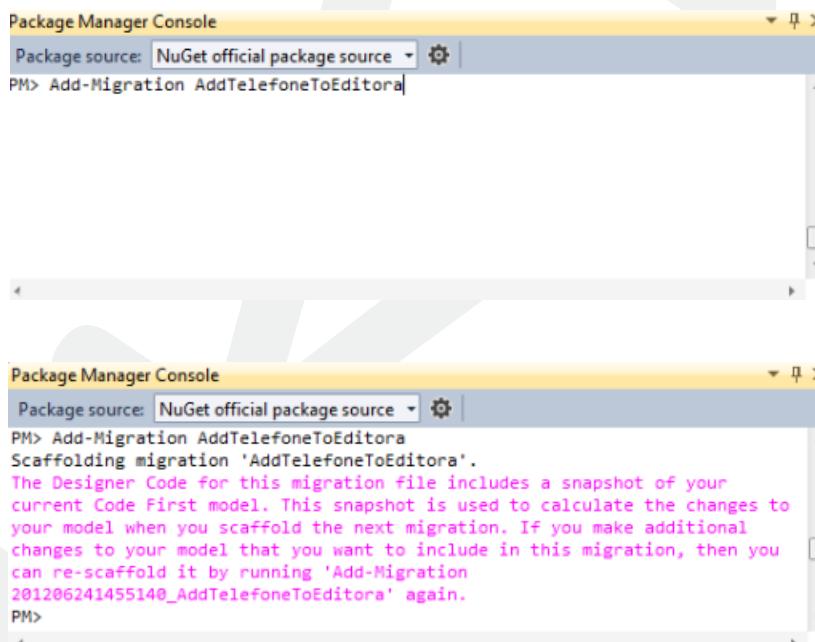




```
Package Manager Console
Package source: NuGet official package source
PM> Enable-Migrations
Checking if the context targets an existing database...
Detected database created with a database initializer. Scaffolded migration
'201206241553487_InitialCreate' corresponding to existing database. To use
an automatic migration instead, delete the Migrations folder and re-run
Enable-Migrations specifying the -EnableAutomaticMigrations parameter.
Code First Migrations enabled for project K19.
PM>
```

Verifique que uma pasta chamada **Migrations** com dois arquivos: <DATA>_InitialCreate.cs e Configuration.cs foi gerada no projeto **CodeFirstMigrations**.

- 9 Após habilitar o Code First Migrations no projeto **CodeFirstMigrations**, devemos adicionar uma migração para atualizar o banco de dados adicionando a coluna Telefone na tabela Editoras. Crie uma migração chamada *AddTelefoneToEditora* através do comando Add-Migration. Execute este comando através do Package Manager Console.



```
Package Manager Console
Package source: NuGet official package source
PM> Add-Migration AddTelefoneToEditora

Package Manager Console
Package source: NuGet official package source
PM> Add-Migration AddTelefoneToEditora
Scaffolding migration 'AddTelefoneToEditora'.
The Designer Code for this migration file includes a snapshot of your
current Code First model. This snapshot is used to calculate the changes to
your model when you scaffold the next migration. If you make additional
changes to your model that you want to include in this migration, then you
can re-scaffold it by running 'Add-Migration
201206241455140_AddTelefoneToEditora' again.
PM>
```

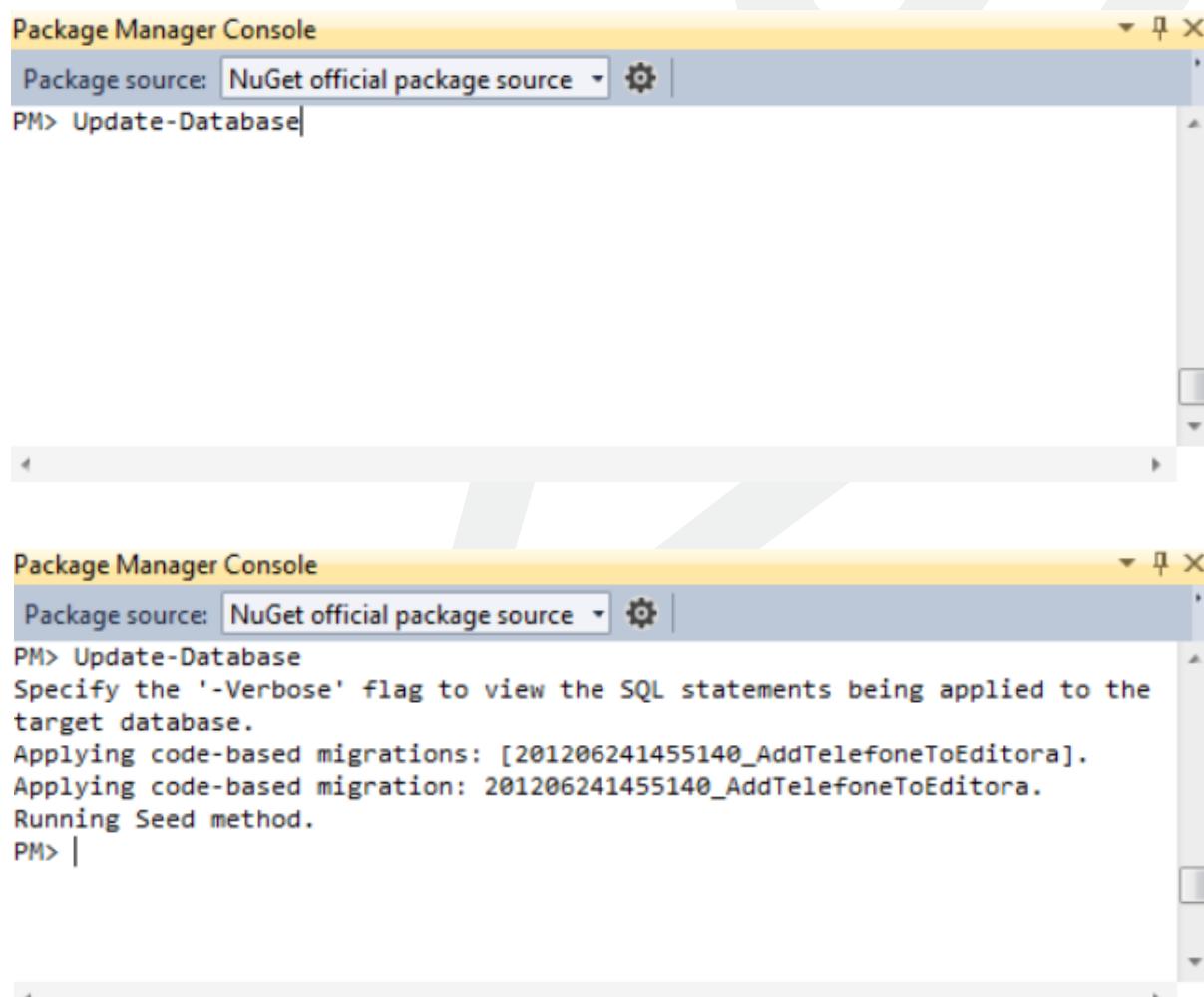
Verifique que um arquivo de migração foi criado na pasta **Migrations**:

```
1 namespace CodeFirstMigrations.Migrations
2 {
3     using System;
4     using System.Data.Entity.Migrations;
5
6     public partial class AddTelefoneToEditora : DbMigration
7     {
8         public override void Up()
9         {
10             AddColumn("dbo.Editoras", "Telefone", c => c.String());
11         }
12     }
```

```
13     public override void Down()
14     {
15         DropIndex("dbo.Editoras", "Telefone");
16     }
17 }
```

Código C# B.12: <DATA>_AddTelefoneToEditora.cs

- 10 Para atualizar a tabela Editoras no banco de dados, utilize o comando Update-Database que é executado através do Package Manager Console.



Verifique que a coluna *Telefone* foi adicionada a tabela **Editoras**.

- 11 Defina uma classe Livro na pasta **Models** conforme o código abaixo:

```
1 namespace CodeFirstMigrations.Models
2 {
3     public class Livro
4     {
5         public int Id { get; set; }
6         public string Titulo { get; set; }
```

```

7     public decimal Preco { get; set; }
8     public int EditoraId { get; set; }
9     public Editora Editora { get; set; }
10    }
11 }
```

Código C# B.13: Livro.cs

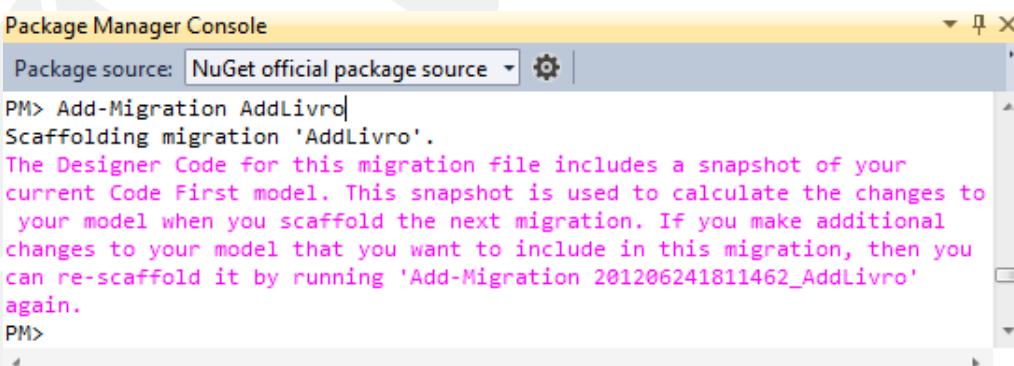
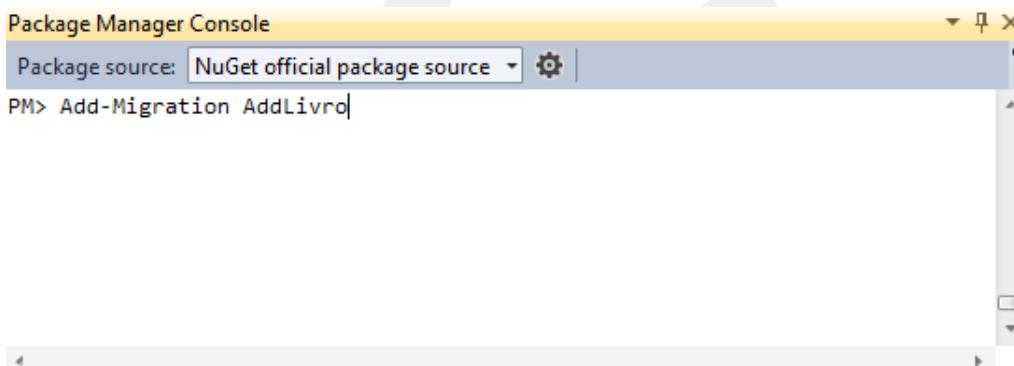
- 12 Altere a classe K19Context para mapear a classe Livro para uma tabela no banco de dados.

```

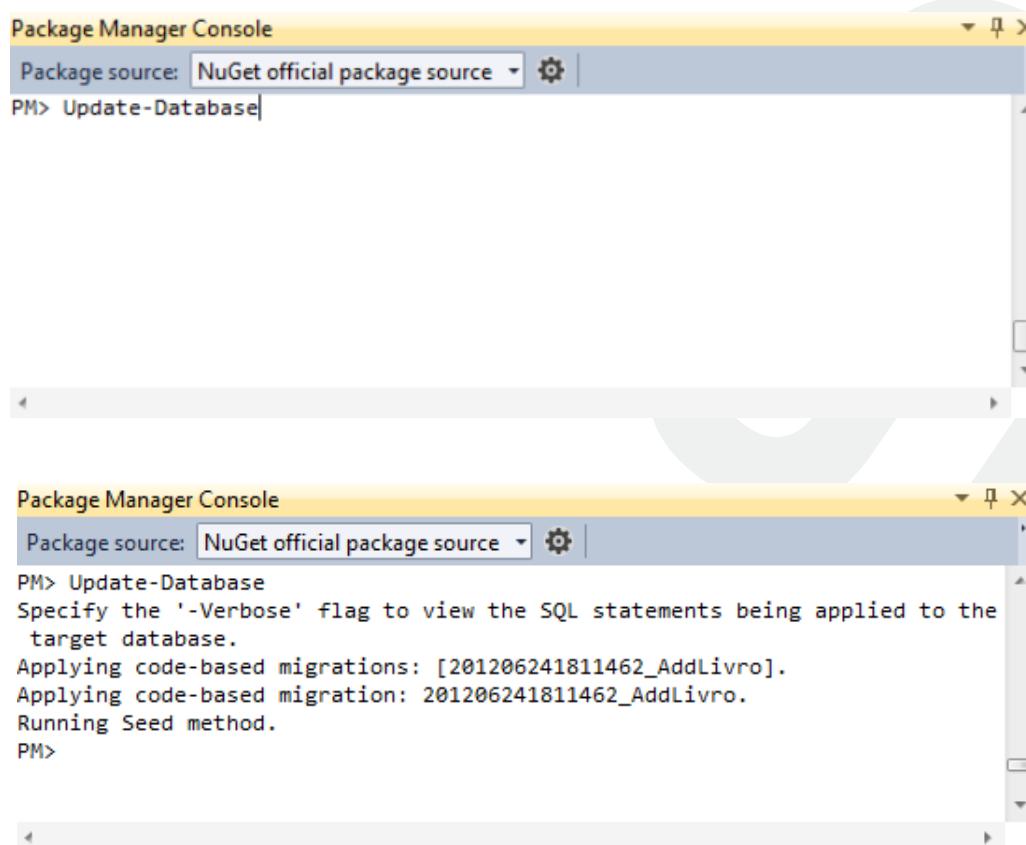
1 using System.Data.Entity;
2
3 namespace CodeFirstMigrations.Models
4 {
5     public class K19Context : DbContext
6     {
7         public DbSet<Editora> Editoras { get; set; }
8         public DbSet<Livro> Livros { get; set; }
9     }
10 }
```

Código C# B.14: K19Context.cs

- 13 Para gerar a tabela Livros no banco de dados, devemos criar uma migração. Crie uma migração chamada AddLivro através do comando Add-Migration.



- 14 Execute o comando Update-Database através do Package Manager Console para gerar a tabela referente a classe Livro no banco de dados.



The image shows two side-by-side screenshots of the Package Manager Console in Visual Studio. Both windows have a title bar 'Package Manager Console' and a dropdown menu 'Package source: NuGet official package source'. The top window shows the command 'PM> Update-Database' entered. The bottom window shows the output of the command:

```
PM> Update-Database
Specify the '-Verbose' flag to view the SQL statements being applied to the
target database.
Applying code-based migrations: [201206241811462_AddLivro].
Applying code-based migration: 201206241811462_AddLivro.
Running Seed method.
PM>
```

Verifique que a tabela referente a classe Livro foi gerada no banco de dados.



PROJETO

Nos capítulos anteriores, vimos os recursos do ASP .NET MVC e do Entity Framework. Agora, vamos solidificar os conhecimentos obtidos e, além disso, mostraremos alguns padrões e conceitos relacionados ao desenvolvimento de aplicações web.

Como exemplo de aplicação desenvolveremos uma aplicação de cadastro de jogadores e seleções de futebol.

Modelo

Por onde começar o desenvolvimento de uma aplicação? Essa é uma questão recorrente. Um ótimo ponto de partida é desenvolver as entidades principais da aplicação. No nosso caso, vamos nos restringir às entidades **Selecao** e **Jogador**. Devemos estabelecer um relacionamento entre essas entidades já que um jogador atua em uma seleção.



Exercícios de Fixação

- 1 Crie um projeto do tipo **ASP .NET MVC 4 Web Application** chamado **K19-CopaDoMundo** seguindo os passos vistos no exercício do capítulo **4**.
- 2 Adicione na pasta **Models** as seguintes classes.

```

1 namespace K19CopaDoMundo.Models
2 {
3     public class Selecao
4     {
5         public int Id { get; set; }
6         public string Pais { get; set; }
7         public string Tecnico { get; set; }
8     }
9 }
```

Código C# C.1: Selecao.cs

```

1 namespace K19CopaDoMundo.Models
2 {
3     public class Jogador
4     {
5         public int Id { get; set; }
6         public string Nome { get; set; }
7         public string Posicao { get; set; }
8         public DateTime Nascimento { get; set; }
9         public double Altura { get; set; }
10        public int SelecaoId { get; set; }
11        public Selecao Selecao { get; set; }
12 }
```

```
12 }  
13 }
```

Código C# C.2: Jogador.cs

Persistência - Mapeamento

Depois de definir algumas entidades podemos começar o processo de implementação da persistência da nossa aplicação. Vamos aplicar os recursos do Entity Framework - Code First que aprendemos nos primeiros capítulos. Inicialmente, vamos definir o mapeamento das nossas entidades através de uma classe derivada de *DbContext* e acrescentar as propriedades referentes a chave primária e chave estrangeira.



Exercícios de Fixação

3

Adicione as seguintes propriedades e anotações as classes Selecao e Jogador.

```
1 using System.ComponentModel.DataAnnotations.Schema;  
2  
3 namespace K19CopaDoMundo.Models  
4 {  
5     [Table("Selecoes")]  
6     public class Selecao  
7     {  
8         public int Id { get; set; }  
9         public string Pais { get; set; }  
10        public string Tecnico { get; set; }  
11        public virtual List<Jogador> Jogadores { get; set; }  
12    }  
13 }
```

Código C# C.3: Selecao.cs

```
1 using System.ComponentModel.DataAnnotations.Schema;  
2  
3 namespace K19CopaDoMundo.Models  
4 {  
5     [Table("Jogadores")]  
6     public class Jogador  
7     {  
8         public int Id { get; set; }  
9         public string Nome { get; set; }  
10        public string Posicao { get; set; }  
11        public DateTime Nascimento { get; set; }  
12        public double Altura { get; set; }  
13        public int SelecaoId { get; set; }  
14        [InverseProperty("Jogadores")]  
15        public virtual Selecao Selecao { get; set; }  
16    }  
17 }
```

Código C# C.4: Jogador.cs

- 4 Adicione a classe K19CopaDoMundoContext a pasta **Models**.

```

1 using System.Data.Entity;
2
3 namespace K19CopaDoMundo.Models
4 {
5     public class K19CopaDoMundoContext : DbContext
6     {
7         public DbSet<Selecao> Selecoes { get; set; }
8         public DbSet<Jogador> Jogadores { get; set; }
9     }
10 }
```

Código C# C.5: K19CopaDoMundoContext.cs

Persistência - Configuração

Precisamos definir a nossa string de conexão para que a nossa aplicação utilize a base de dados *k19copadomundo* como padrão.



Exercícios de Fixação

- 5 Acrescente ao arquivo **Web.config**, que fica na raiz do projeto, a string de conexão.

```

1 <connectionStrings>
2     <add
3         name="K19CopaDoMundoContext" providerName="System.Data.SqlClient"
4         connectionString="Server=.\\SQLEXPRESS;Database=k19copadomundo;
5         User Id=sa; Password=sa;Trusted_Connection=False;MultipleActiveResultSets=True" />
6 
```

Código XML C.1: Web.config

Persistência - Reppositórios

Vamos deixar os repositórios para acessar as entidades da nossa aplicação preparados. Os repositórios precisam de DbContexts para realizar as operações de persistência. Então, cada repositório terá um construtor para receber um DbContext como parâmetro.



Exercícios de Fixação

- 6 Crie uma classe na pasta **Models** chamada **SelecaoRepository**.

```

1 using System;
2 using System.Collections.Generic;
3
4 namespace K19CopaDoMundo.Models
5 {
6     public class SelecaoRepository : IDisposable
7     {
8         private bool disposed = false;
```

```
9     private K19CopaDoMundoContext context;
10
11    public SelecaoRepository(K19CopaDoMundoContext context)
12    {
13        this.context = context;
14    }
15
16
17    public void Adiciona(Selecao selecao)
18    {
19        context.Selecoes.Add(selecao);
20    }
21
22    public List<Selecao> Selecoes
23    {
24        get
25        {
26            return context.Selecoes.ToList();
27        }
28    }
29    public void Salva()
30    {
31        context.SaveChanges();
32    }
33
34    protected virtual void Dispose(bool disposing)
35    {
36        if (!this.disposed)
37        {
38            if (disposing)
39            {
40                context.Dispose();
41            }
42        }
43        this.disposed = true;
44    }
45
46    public void Dispose()
47    {
48        Dispose(true);
49        GC.SuppressFinalize(this);
50    }
51}
52}
```

Código C# C.6: SelecaoRepository.cs

- 7 Analogamente crie um repositório de jogadores.

```
1 using System;
2 using System.Collections.Generic;
3
4 namespace K19CopaDoMundo.Models
5 {
6     public class JogadorRepository : IDisposable
7     {
8         private bool disposed = false;
9         private K19CopaDoMundoContext context;
10
11         public JogadorRepository(K19CopaDoMundoContext context)
12         {
13             this.context = context;
14         }
15
16         public void Adiciona(Jogador jogador)
17         {
18             context.Jogadores.Add(jogador);
19         }
20
21         protected virtual void Dispose(bool disposing)
22         {
23             if (!this.disposed)
24             {
25                 if (disposing)
26                 {
27                     context.Dispose();
28                 }
29             }
30             this.disposed = true;
31         }
32
33         public void Dispose()
34         {
35             Dispose(true);
36             GC.SuppressFinalize(this);
37         }
38     }
39 }
```

```

19 }
20
21     public List<Jogador> Jogadores
22     {
23         get { return context.Jogadores.ToList(); }
24     }
25
26     public void Salva()
27     {
28         context.SaveChanges();
29     }
30
31     protected virtual void Dispose(bool disposing)
32     {
33         if (!this.disposed)
34         {
35             if (disposing)
36             {
37                 context.Dispose();
38             }
39         }
40         this.disposed = true;
41     }
42
43     public void Dispose()
44     {
45         Dispose(true);
46         GC.SuppressFinalize(this);
47     }
48 }
49

```

Código C# C.7: JogadorRepository.cs

Unit of Work

O único propósito de criar uma classe **UnitOfWork** é ter certeza que quando temos múltiplos repositórios eles compartilham o mesmo *DbContext*. Para isto, devemos apenas criar um método **Salva** e uma propriedade para cada repositório.



Exercícios de Fixação

- 8 Crie uma classe **UnitOfWork** na pasta **Models**.

```

1 using System;
2 using System.Collections.Generic;
3
4 namespace K19CopaDoMundo.Models
5 {
6     public class UnitOfWork : IDisposable
7     {
8         private bool disposed = false;
9         private K19CopaDoMundoContext context = new K19CopaDoMundoContext();
10        private SelecaoRepository selecaoRepository;
11        private JogadorRepository jogadorRepository;
12
13        public JogadorRepository JogadorRepository
14        {
15            get
16            {
17

```

```
18     if (jogadorRepository == null)
19     {
20         jogadorRepository = new JogadorRepository(context);
21     }
22     return jogadorRepository;
23 }
24 }
25
26 public SelecaoRepository SelecaoRepository
27 {
28     get
29     {
30         if (selecaoRepository == null)
31         {
32             selecaoRepository = new SelecaoRepository(context);
33         }
34         return selecaoRepository;
35     }
36 }
37
38 public void Salva()
39 {
40     context.SaveChanges();
41 }
42
43 protected virtual void Dispose(bool disposing)
44 {
45     if (!this.disposed)
46     {
47         if (disposing)
48         {
49             context.Dispose();
50         }
51     }
52     this.disposed = true;
53 }
54
55 public void Dispose()
56 {
57     Dispose(true);
58     GC.SuppressFinalize(this);
59 }
60 }
61 }
```

Código C# C.8: UnitOfWork.cs

Apresentação - Template

Vamos definir um template para as telas da nossa aplicação. Aplicaremos algumas regras CSS para melhorar a parte visual das telas.



Exercícios de Fixação

- 9 Na pasta **Content**, altere o arquivo **Site.css** acrescentando algumas regras CSS.

```
1 .logo
2 {
3     vertical-align: middle;
4 }
```

```
6 .botao
7 {
8     background-color: #064D83;
9     margin: 0 0 0 20px;
10    color: white;
11    text-decoration: none;
12    font-size: 20px;
13    line-height: 20px;
14    padding: 5px;
15    vertical-align: middle;
16 }
17
18 .botao:hover
19 {
20     background-color: #cccccc;
21     color: #666666;
22 }
23
24 .formulario fieldset
25 {
26     float: left;
27     margin: 0 0 20px 0;
28     border: 1px solid #333333;
29 }
30
31 .formulario fieldset legend
32 {
33     color: #064D83;
34     font-weight: bold;
35 }
36
37 .botao-formulario
38 {
39     background-color: #064D83;
40     color: #ffffff;
41     padding: 5px;
42     vertical-align: middle;
43     border: none;
44 }
45
46 .titulo
47 {
48     color: #064D83;
49     clear: both;
50 }
51
52 .tabela
53 {
54     border: 1px solid #064D83;
55     border-collapse: collapse;
56 }
57
58 .tabela tr th
59 {
60     background-color: #064D83;
61     color: #ffffff;
62 }
63
64 .tabela tr th, .tabela tr td
65 {
66     border: 1px solid #064D83;
67     padding: 2px 5px;
68 }
69
70
71 /* Styles for validation helpers
72 -----
73 .field-validation-error
74 {
75     color: #ff0000;
```

```
76 }
77 .field-validation-valid
78 {
79     display: none;
80 }
81
82 .input-validation-error
83 {
84     border: 1px solid #ff0000;
85     background-color: #ffeeee;
86 }
87
88 .validation-summary-errors
89 {
90     font-weight: bold;
91     color: #ff0000;
92 }
93
94 .validation-summary-valid
95 {
96     display: none;
97 }
98 }
```

Código CSS C.1: Site.css

- 10 Copie o arquivo **k19-logo.png** da pasta **K19-Arquivos** da sua Área de Trabalho para a pasta **Images**.

- 11 Agora altere o arquivo **_Layout.cshtml**.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="utf-8" />
5     <title>Copa do Mundo</title>
6     <link href="~/__TemplateIcon.ico" rel="shortcut icon" type="image/x-icon" />
7     <meta name="viewport" content="width=device-width" />
8     @Styles.Render("~/Content/themes/base/css", "~/Content/css")
9     @Scripts.Render("~/bundles/modernizr")
10 </head>
11 <body>
12     <header>
13         <div class="content-wrapper">
14             <div class="float-left">
15                 <p class="site-title"></p> </div>
17             <div class="float-right">
18                 <nav>
19                     <ul id="menu">
20                         <li>@Html.ActionLink("Selecoes", "Index", "Selecao", null, new { @class = "botao" })</li>
21                         <li>@Html.ActionLink("Jogadores", "Index", "Jogador", null, new { @class = "botao" })</li>
22                     </ul>
23                     </nav>
24                 </div>
25             </div>
26         </header>
27         <div id="body">
28             @RenderSection("featured", required: false)
29             <section class="content-wrapper main-content clear-fix">
30                 @RenderBody()
31             </section>
32         </div>
```

```

33 <footer>
34     <div class="content-wrapper">
35         <div class="float-left">
36             <p>&copy; @DateTime.Now.Year - K19 Copa do Mundo</p>
37         </div>
38         <div class="float-right">
39             <ul id="social">
40                 <li><a href="http://facebook.com/k19treinamentos" class="facebook">Facebook - K19 Treinamentos</a></li>
41                 <li><a href="http://twitter.com/k19treinamentos" class="twitter">Twitter - K19 Treinamentos</a></li>
42             </ul>
43         </div>
44     </div>
45 </footer>
46 @Scripts.Render("~/bundles/jquery")
47 @RenderSection("scripts", required: false)
48 </body>
49 </html>

```

Código CSHTML C.1: _Layout.cshtml

Cadastrando e Listando Seleções

Na tela de seleções, vamos adicionar um formulário para cadastrar novas seleções e uma tabela para apresentar as já cadastradas. Aplicaremos regras de validação específicas para garantir que nenhum dado incorreto seja armazenado no banco de dados.



Exercícios de Fixação

- 12 Para cadastrar a seleção, devemos definir o controlador.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Data;
4  using System.Data.Entity;
5  using System.Linq;
6  using System.Web;
7  using System.Web.Mvc;
8  using K19CopaDoMundo.Models;
9
10 namespace K19CopaDoMundo.Controllers
11 {
12     public class SelecaoController : Controller
13     {
14         private UnitOfWork unitOfWork = new UnitOfWork();
15
16         public ActionResult Create()
17         {
18             return View();
19         }
20
21         protected override void Dispose(bool disposing)
22         {
23             unitOfWork.Dispose();
24             base.Dispose(disposing);
25         }
26     }
27 }

```

Código C# C.9: SelecaoController.cs

- 13 Vamos criar uma tela **Create.cshtml** para cadastrar as seleções. Adicione o arquivo a pasta **Views/Selecoes** com o seguinte conteúdo.

```

1 @model K19CopaDoMundo.Models.Selecao
2 @{
3     ViewBag.Title = "Create";
4 }
5
6 <h2>Create</h2>
7
8 @using (Html.BeginForm()) {
9     @Html.ValidationSummary(true)
10
11     <fieldset>
12         <legend>Selecao</legend>
13
14         <div class="editor-label">
15             @Html.LabelFor(model => model.Pais)
16         </div>
17         <div class="editor-field">
18             @Html.EditorFor(model => model.Pais)
19             @Html.ValidationMessageFor(model => model.Pais)
20         </div>
21
22         <div class="editor-label">
23             @Html.LabelFor(model => model.Tecnico)
24         </div>
25         <div class="editor-field">
26             @Html.EditorFor(model => model.Tecnico)
27             @Html.ValidationMessageFor(model => model.Tecnico)
28         </div>
29
30         <p>
31             <input type="submit" value="Create" />
32         </p>
33     </fieldset>
34 }
35
36 <div>
37     @Html.ActionLink("Listagem de Seleções", "Index")
38 </div>
39
40 @section Scripts {
41     @Scripts.Render("~/bundles/jqueryval")
42 }
43 
```

Código CSHTML C.2: Create.cshtml

- 14 O próximo passo é definir a action que irá salvar a seleção no nosso banco de dados. Devemos também acrescentar as validações a nossa entidade.

```

1 [HttpPost]
2 public ActionResult Create(Selecao selecao)
3 {
4     if (ModelState.IsValid)
5     {
6         unitOfWork.SelecaoRepository.Adiciona(selecao);
7         unitOfWork.Salva();
8         return RedirectToAction("Index");
9     }
10 } 
```

```

9     }
10    return View(selecao);
11 }
```

Código C# C.10: SelecaoController.cs

```

1 using System.ComponentModel.DataAnnotations.Schema;
2 using System.ComponentModel.DataAnnotations;
3
4 namespace K19CopaDoMundo.Models
5 {
6     [Table("Selecoes")]
7     public class Selecao
8     {
9         public int Id { get; set; }
10        [Required(ErrorMessage = "O campo País é obrigatório.")]  
public string Pais { get; set; }
11        [Required(ErrorMessage = "O campo Técnico é  
obrigatório.")] public string Tecnico { get; set; }
12        public virtual List<Jogador> Jogadores { get; set; }
13    }
14 }
```

Código C# C.11: Selecao.cs

- 15** Defina a action e a página para listar todas as entidades de seleção.

```

1 public ActionResult Index()
2 {
3     return View(unitOfWork.SelecaoRepository.Selecoes);
4 }
```

Código C# C.12: SelecaoController.cs

```

1 @model IEnumerable<K19CopaDoMundo.Models.Selecao>
2
3 @{
4     ViewBag.Title = "Index";
5 }
6
7 <h2>Index</h2>
8
9 <p>
10    @Html.ActionLink("Create New", "Create")
11 </p>
12 <table>
13     <tr>
14         <th>
15             @Html.DisplayNameFor(model => model.Pais)
16         </th>
17         <th>
18             @Html.DisplayNameFor(model => model.Tecnico)
19         </th>
20         <th></th>
21     </tr>
22
23 @foreach (var item in Model) {
24     <tr>
25         <td>
26             @Html.DisplayFor(modelItem => item.Pais)
27         </td>
28         <td>
29             @Html.DisplayFor(modelItem => item.Tecnico)
30         </td>
31     </tr>
```

```
32  }
33  </table>
34 
```

Código CSHTML C.3: Index.cshtml

- 16 Vamos definir a tela de listagem de Seleções como a página principal do nosso site. Altere a rota padrão no arquivo **RouteConfig.cs**.

```
1 routes.MapRoute(
2     name: "Default",
3     url: "{controller}/{action}/{id}",
4     defaults: new { controller = "Selecao", action = "Index",
5     id = UrlParameter.Optional } );
```

Código C# C.13: RouteConfig.cs

Removendo Seleções

Vamos acrescentar a funcionalidade de remover seleções.



Exercícios de Fixação

17

Acrescente uma coluna na tabela de listagem de seleções.

```
1 @model IEnumerable<K19CopaDoMundo.Models.Selecao>
2
3 @{
4     ViewBag.Title = "Index";
5 }
6
7 <h2>Index</h2>
8
9 <p>
10    @Html.ActionLink("Create New", "Create")
11 </p>
12 <table>
13     <tr>
14         <th>
15             @Html.DisplayNameFor(model => model.Pais)
16         </th>
17         <th>
18             @Html.DisplayNameFor(model => model.Tecnico)
19         </th>
20         <th></th>
21     </tr>
22
23 @foreach (var item in Model) {
24     <tr>
25         <td>
26             @Html.DisplayFor(modelItem => item.Pais)
27         </td>
28         <td>
29             @Html.DisplayFor(modelItem => item.Tecnico)
30         </td>
31         <td>
```

```

32     @Html.ActionLink("Remover", "Delete", new {id = item.Id})
33   </td>
34   </tr>
35 }
36
37 </table>

```

Código CSHTML C.4: Index.cshtml

- 18 Defina um método **Busca** na classe **SelecaoRepository** que retorna uma entidade seleção a partir de um parâmetro **id**.

```

1 public Selecao Busca(int id)
2 {
3     return context.Selecoes.Find(id);
4 }

```

Código C# C.14: SelecaoRepository.cs

- 19 Defina uma action **Delete** que irá mostrar a tela de confirmação de remoção da entidade.

```

1 public ActionResult Delete(int id)
2 {
3     Selecao selecao = unitOfWork.SelecaoRepository.Busca(id);
4     return View(selecao);
5 }

```

Código C# C.15: SelecaoController.cs

- 20 Defina a tela de confirmação de remoção da seleção.

```

1 @model K19CopaDoMundo.Models.Selecao
2
3 @{
4     ViewBag.Title = "Delete";
5 }
6
7 <h2>Remoção de Seleção</h2>
8
9 <h3>Você tem certeza que deseja remover esta seleção?</h3>
10 <fieldset>
11     <legend>Selecao</legend>
12
13     <div class="display-label">
14         @Html.DisplayNameFor(model => model.Pais)
15     </div>
16     <div class="display-field">
17         @Html.DisplayFor(model => model.Pais)
18     </div>
19
20     <div class="display-label">
21         @Html.DisplayNameFor(model => model.Tecnico)
22     </div>
23     <div class="display-field">
24         @Html.DisplayFor(model => model.Tecnico)
25     </div>
26 </fieldset>
27 @using (Html.BeginForm()) {
28     <p>
29         <input type="submit" value="Delete" /> |
30         @Html.ActionLink("Listagem De Seleções", "Index")

```

```
31     </p>
32 }
```

Código CSHTML C.5: Delete.cshtml

- 21 Defina um método na classe **SelecaoRepository** que remove uma entidade seleção a partir de um parâmetro *id*.

```
1 public void Remove(int id)
2 {
3     Selecao selecao = Busca(id);
4     context.Selecoes.Remove(selecao);
5 }
```

Código C# C.16: SelecaoRepository.cs

- 22 Defina a action que remove a seleção do banco de dados.

```
1 [HttpPost]
2 [ActionName("Delete")]
3 public ActionResult DeleteConfirmed(int id)
4 {
5     unitOfWork.SelecaoRepository.Remove(id);
6     unitOfWork.Salva();
7     return RedirectToAction("Index");
8 }
```

Código C# C.17: SelecoesController.cs

Cadastrando, Listando e Removendo Jogadores

Na tela de jogadores, vamos adicionar um formulário para cadastrar novos jogadores e uma tabela para apresentar os já cadastrados. Aplicaremos regras de validação específicas para garantir que nenhum dado incorreto seja armazenado no banco de dados.



Exercícios de Fixação

- 23 Para cadastrar o jogador, devemos definir o controlador.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Data;
4 using System.Data.Entity;
5 using System.Linq;
6 using System.Web;
7 using System.Web.Mvc;
8 using K19CopaDoMundo.Models;
9
10 namespace K19CopaDoMundo.Controllers
11 {
12     public class JogadorController : Controller
13     {
14         private UnitOfWork unitOfWork = new UnitOfWork();
```

```

16     public ActionResult Create()
17     {
18         ViewBag.SelecaoId = new SelectList(unitOfWork.SelecaoRepository.Selecoes, "Id", ←
19             "País");
20         return View();
21     }
22
23     protected override void Dispose(bool disposing)
24     {
25         unitOfWork.Dispose();
26         base.Dispose(disposing);
27     }
28 }
```

Código C# C.18: JogadorController.cs

- 24 Vamos criar uma tela **Create.cshtml** para cadastrar os jogadores. Adicione o arquivo a pasta **Views/Jogador** com o seguinte conteúdo.

```

1 @model K19CopaDoMundo.Models.Jogador
2
3 @{
4     ViewBag.Title = "Create";
5 }
6
7 <h2>Create</h2>
8
9 @using (Html.BeginForm()) {
10     @Html.ValidationSummary(true)
11
12     <fieldset>
13         <legend>Jogador</legend>
14
15         <div class="editor-label">
16             @Html.LabelFor(model => model.Nome)
17         </div>
18         <div class="editor-field">
19             @Html.EditorFor(model => model.Nome)
20             @Html.ValidationMessageFor(model => model.Nome)
21         </div>
22
23         <div class="editor-label">
24             @Html.LabelFor(model => model.Posicao)
25         </div>
26         <div class="editor-field">
27             @Html.EditorFor(model => model.Posicao)
28             @Html.ValidationMessageFor(model => model.Posicao)
29         </div>
23
31         <div class="editor-label">
32             @Html.LabelFor(model => model.Nascimento)
33         </div>
34         <div class="editor-field">
35             @Html.EditorFor(model => model.Nascimento)
36             @Html.ValidationMessageFor(model => model.Nascimento)
37         </div>
38
39         <div class="editor-label">
40             @Html.LabelFor(model => model.Altura)
41         </div>
42         <div class="editor-field">
43             @Html.EditorFor(model => model.Altura)
44             @Html.ValidationMessageFor(model => model.Altura)
45         </div>
46
47         <div class="editor-label">
```

```

48         @Html.LabelFor(model => model.SelecaoId)
49     </div>
50     <div class="editor-field">
51         @Html.DropDownList("SelecaoId", String.Empty)
52         @Html.ValidationMessageFor(model => model.SelecaoId)
53     </div>
54
55     <p>
56         <input type="submit" value="Create" />
57     </p>
58 </fieldset>
59 }
60
61 <div>
62     @Html.ActionLink("Listagem de Jogadores", "Index")
63 </div>
64
65 @section Scripts {
66     @Scripts.Render("~/bundles/jqueryval")
67 }
```

Código CSHTML C.6: Create.cshtml

25 O próximo passo é definir a action que irá salvar o jogador no nosso banco de dados. Devemos também acrescentar as validações a nossa entidade.

```

1 [HttpPost]
2     public ActionResult Create(Jogador jogador)
3     {
4         if (ModelState.IsValid)
5         {
6             unitOfWork.JogadorRepository.Adiciona(jogador);
7             unitOfWork.Salva();
8             return RedirectToAction("Index");
9         }
10        ViewBag.Selecaoid = new SelectList(unitOfWork.SelecaoRepository.Selecoes, ←
11            "Id", "Pais");
12        return View();
13    }
```

Código C# C.19: JogadorController.cs

```

1 using System.ComponentModel.DataAnnotations.Schema;
2 using System.ComponentModel.DataAnnotations;
3
4 namespace K19CopaDoMundo.Models
5 {
6     [Table("Jogadores")]
7     public class Jogador
8     {
9         public int Id { get; set; }
10        [Required(ErrorMessage = "O campo Nome é obrigatório.")]
11        public string Nome { get; set; }
12        [Required(ErrorMessage = "O campo Posicao é
13            obrigatório.")] public string Posicao { get; set; }
14        [Required(ErrorMessage = "O campo Nascimento é
15            obrigatório.")]
16        [DataType(DataType.Date)]
17        public DateTime Nascimento { get; set; }
18        [Required(ErrorMessage = "O campo Altura é
19            obrigatório.")]
20        public double Altura { get; set; }
21        public int SelecaoId { get; set; }
22        [InverseProperty("Jogadores")]
23        public virtual Selecao Selecao { get; set; }
24    }
```

25 }

Código C# C.20: Jogador.cs

- 26** Defina a action e a página para listar todas as entidades de jogador.

```
1 public ActionResult Index()
2 {
3     return View(unitOfWork.JogadorRepository.Jogadores);
4 }
```

Código C# C.21: JogadorController.cs

```
1 @model IEnumerable<K19CopaDoMundo.Models.Jogador>
2 @{
3     ViewBag.Title = "Listagem de Jogadores";
4 }
5
6 <h2>Listagem de Jogadores</h2>
7
8 <p>
9     @Html.ActionLink("Cadastrar Jogador", "Create")
10 </p>
11 <table>
12     <tr>
13         <th>
14             @Html.DisplayNameFor(model => model.Nome)
15         </th>
16         <th>
17             @Html.DisplayNameFor(model => model.Posicao)
18         </th>
19         <th>
20             @Html.DisplayNameFor(model => model.Nascimento)
21         </th>
22         <th>
23             @Html.DisplayNameFor(model => model.Altura)
24         </th>
25         <th>
26             @Html.DisplayNameFor(model => model.SelecaoId)
27         </th>
28         <th></th>
29     </tr>
30
31 @foreach (var item in Model) {
32     <tr>
33         <td>
34             @Html.DisplayFor(modelItem => item.Nome)
35         </td>
36         <td>
37             @Html.DisplayFor(modelItem => item.Posicao)
38         </td>
39         <td>
40             @Html.DisplayFor(modelItem => item.Nascimento)
41         </td>
42         <td>
43             @Html.DisplayFor(modelItem => item.Altura)
44         </td>
45         <td>
46             @Html.DisplayFor(modelItem => item.SelecaoId)
47         </td>
48     </tr>
49 }
50 </table>
```

Código CSHTML C.7: Index.cshtml

Removendo Jogadores

Vamos acrescentar a funcionalidade de remover jogadores.



Exercícios de Fixação

- 27 Acrescente uma coluna na tabela de listagem de jogadores.

```
1 @model IEnumerable<K19CopaDoMundo.Models.Jogador>
2
3 @{
4     ViewBag.Title = "Listagem de Jogadores";
5 }
6
7 <h2>Listagem de Jogadores</h2>
8
9 <p>
10    @Html.ActionLink("Cadastrar Jogador", "Create")
11 </p>
12 <table>
13     <tr>
14         <th>
15             @Html.DisplayNameFor(model => model.Nome)
16         </th>
17         <th>
18             @Html.DisplayNameFor(model => model.Posicao)
19         </th>
20         <th>
21             @Html.DisplayNameFor(model => model.Nascimento)
22         </th>
23         <th>
24             @Html.DisplayNameFor(model => model.Altura)
25         </th>
26         <th>
27             @Html.DisplayNameFor(model => model.SelecaoId)
28         </th>
29         <th></th>
30     </tr>
31
32 @foreach (var item in Model) {
33     <tr>
34         <td>
35             @Html.DisplayFor(modelItem => item.Nome)
36         </td>
37         <td>
38             @Html.DisplayFor(modelItem => item.Posicao)
39         </td>
40         <td>
41             @Html.DisplayFor(modelItem => item.Nascimento)
42         </td>
43         <td>
44             @Html.DisplayFor(modelItem => item.Altura)
45         </td>
46         <td>
47             @Html.DisplayFor(modelItem => item.SelecaoId)
48         </td>
49         <td>@Html.ActionLink("Remover", "Delete",
50             new{id=item.Id})</td> </tr>
```

```
51    }
52  </table>
```

Código CSHTML C.8: Index.cshtml

- 28 Defina um método **Busca** na classe **JogadorRepository** que retorna uma entidade jogador a partir de um parâmetro **id**.

```
1 public Jogador Busca(int id)
2 {
3     return context.Jogadores.Find(id);
4 }
```

Código C# C.22: JogadorRepository.cs

- 29 Defina uma action **Delete** que irá mostrar a tela de confirmação de remoção da entidade.

```
1 public ActionResult Delete(int id)
2 {
3     Jogador jogador = unitOfWork.JogadorRepository.Busca(id);
4     return View(jogador);
5 }
```

Código C# C.23: JogadorController.cs

- 30 Defina a tela de confirmação de remoção do jogador.

```
1 @model K19CopaDoMundo.Models.Jogador
2 @{
3     ViewBag.Title = "Remoção do Jogador";
4 }
5
6
7 <h2>Remoção do Jogador</h2>
8
9 <h3>Você tem certeza que deseja remover este jogador?</h3>
10 <fieldset>
11     <legend>Jogador</legend>
12
13     <div class="display-label">
14         @Html.DisplayNameFor(model => model.Nome)
15     </div>
16     <div class="display-field">
17         @Html.DisplayFor(model => model.Nome)
18     </div>
19
20     <div class="display-label">
21         @Html.DisplayNameFor(model => model.Posicao)
22     </div>
23     <div class="display-field">
24         @Html.DisplayFor(model => model.Posicao)
25     </div>
26
27     <div class="display-label">
28         @Html.DisplayNameFor(model => model.Nascimento)
29     </div>
30     <div class="display-field">
31         @Html.DisplayFor(model => model.Nascimento)
32     </div>
33
34     <div class="display-label">
```

```

35     @Html.DisplayNameFor(model => model.Altura)
36   </div>
37   <div class="display-field">
38     @Html.DisplayFor(model => model.Altura)
39   </div>
40
41   <div class="display-label">
42     @Html.DisplayNameFor(model => model.SelecaoId)
43   </div>
44   <div class="display-field">
45     @Html.DisplayFor(model => model.SelecaoId)
46   </div>
47 </fieldset>
48 @using (Html.BeginForm()) {
49   <p>
50     <input type="submit" value="Delete" /> |
51     @Html.ActionLink("Listagem de Jogadores", "Index")
52   </p>
53 }

```

Código CSHTML C.9: Delete.cshtml

- 31 Defina um método na classe **JogadorRepository** que remove uma entidade jogador a partir de um parâmetro *id*.

```

1 public void Remove(int id)
2 {
3     Jogador jogador = context.Jogadores.Find(id);
4     context.Jogadores.Remove(jogador);
5 }

```

Código C# C.24: JogadorRepository.cs

- 32 Defina a action que remove o jogador do banco de dados.

```

1 [HttpPost]
2 [ActionName("Delete")]
3 public ActionResult DeleteConfirmed(int id)
4 {
5     unitOfWork.JogadorRepository.Remove(id);
6     unitOfWork.Salva();
7     return RedirectToAction("Index");
8 }

```

Código C# C.25: JogadoresController.cs

Membership e Autorização

Na maioria dos casos, as aplicações devem controlar o acesso dos usuários. Vamos implementar um mecanismo de autenticação na nossa aplicação utilizando filtro e Membership. As requisições feitas pelos usuários passarão pelo filtro. A função do filtro é verificar se o usuário está logado ou não. Se estiver logado o filtro autoriza o acesso. Caso contrário, o filtro redirecionará o usuário para a tela de login.



Exercícios de Fixação

- 33 Adicione a seguinte classe a pasta **Models**:

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web.Mvc;
5  using System.ComponentModel.DataAnnotations;
6
7  namespace K19CopaDoMundo.Models
8  {
9
10     public class ChangePasswordModel
11     {
12         [Required]
13         [DataType(DataType.Password)]
14         [Display(Name = "Senha")]
15         public string OldPassword { get; set; }
16
17         [Required]
18         [StringLength(100, ErrorMessage = "O {0} deve ter no mínimo {2} caracteres.", ←
19                         MinimumLength = 6)]
20         [DataType(DataType.Password)]
21         [Display(Name = "Nova senha")]
22         public string NewPassword { get; set; }
23
24         [DataType(DataType.Password)]
25         [Display(Name = "Confirmação de senha")]
26         [Compare("NewPassword", ErrorMessage = "A senha e a confirmação não conferem."←
27                         )]
28         public string ConfirmPassword { get; set; }
29     }
30
31     public class LoginModel
32     {
33         [Required]
34         [Display(Name = "Usuário")]
35         public string UserName { get; set; }
36
37         [Required]
38         [DataType(DataType.Password)]
39         [Display(Name = "Senha")]
40         public string Password { get; set; }
41
42         [Display(Name = "Lembrar?")]
43         public bool RememberMe { get; set; }
44     }
45
46     public class RegisterModel
47     {
48         [Required]
49         [Display(Name = "Usuário")]
50         public string UserName { get; set; }
51
52         [Required]
53         [DataType(DataType.EmailAddress)]
54         [Display(Name = "Email")]
55         public string Email { get; set; }
56
57         [Required]
58         [StringLength(100, ErrorMessage = "O {0} deve ter no mínimo {2} caracteres.", ←
59                         MinimumLength = 6)]
60         [DataType(DataType.Password)]
61         [Display(Name = "Senha")]
62         public string Password { get; set; }
63
64         [DataType(DataType.Password)]
65         [Display(Name = "Confirmação de senha")]
66         [Compare("Password", ErrorMessage = "A senha e a confirmação não conferem.")]
67         public string ConfirmPassword { get; set; }
68     }
69 }
```

```
66 }  
67 }
```

Código C# C.26: Usuario.cs

34 Acrescente a seguinte classe a pasta **Controllers**.

```
1  using System;  
2  using System.Collections.Generic;  
3  using System.Linq;  
4  using System.Web;  
5  using System.Web.Mvc;  
6  using System.Web.Security;  
7  using K19CopaDoMundo.Models;  
8  
9  namespace K19CopaDoMundo.Controllers  
10 {  
11     public class UsuarioController : Controller  
12     {  
13         //  
14         // GET: /Usuario/Login  
15  
16         public ActionResult Login()  
17         {  
18             return View();  
19         }  
20  
21         //  
22         // POST: /Usuario/Login  
23  
24         [HttpPost]  
25         public ActionResult Login(LoginModel model, string returnUrl)  
26         {  
27             if (ModelState.IsValid)  
28             {  
29                 if (Membership.ValidateUser(model.UserName, model.Password))  
30                 {  
31                     FormsAuthentication.SetAuthCookie(model.UserName, model.RememberMe);  
32                     if (Url.IsLocalUrl(returnUrl) && returnUrl.Length > 1 && returnUrl.  
33                         StartsWith("/")  
34                         && !returnUrl.StartsWith("//") && !returnUrl.StartsWith("//\\\""))  
35                     {  
36                         return RedirectToAction("Index", "Selecao");  
37                     }  
38                     else  
39                     {  
40                         return RedirectToAction("Index", "Selecao");  
41                     }  
42                 }  
43                 else  
44                 {  
45                     ModelState.AddModelError("", "O usuário e/ou a senha está incorreto.");  
46                 }  
47             }  
48  
49             return View(model);  
50         }  
51  
52         //  
53         // GET: /Usuario/LogOff  
54  
55         public ActionResult LogOff()  
56         {  
57             FormsAuthentication.SignOut();  
58  
59             return Redirect("/");  
60         }  
61     }  
62 }
```

```

60
61
62
63 // GET: /Usuario/Register
64
65     public ActionResult Register()
66     {
67         return View();
68     }
69
70
71 // POST: /Usuario/Register
72
73 [HttpPost]
74     public ActionResult Register(RegisterModel model)
75     {
76         if (ModelState.IsValid)
77         {
78             // Attempt to register the user
79             MembershipCreateStatus createStatus;
80             Membership.CreateUser(model.UserName, model.Password, model.Email, null, null, ←
81             true, null, out createStatus);
82
83             if (createStatus == MembershipCreateStatus.Success)
84             {
85                 FormsAuthentication.SetAuthCookie(model.UserName, false /* ←
86                     createPersistentCookie */);
87                 return Redirect("/");
88             }
89             else
90             {
91                 ModelState.AddModelError("", ErrorCodeToString(createStatus));
92             }
93         }
94
95         return View(model);
96     }
97
98 // GET: /Usuario/ChangePassword
99
100 [Authorize]
101     public ActionResult ChangePassword()
102     {
103         return View();
104     }
105
106
107 // POST: /Usuario/ChangePassword
108
109 [Authorize]
110 [HttpPost]
111     public ActionResult ChangePassword(ChangePasswordModel model)
112     {
113         if (ModelState.IsValid)
114         {
115
116             bool changePasswordSucceeded;
117             try
118             {
119                 MembershipUser currentUser = Membership.GetUser(User.Identity.Name, true /* ←
120                     userIsOnline */);
121                 changePasswordSucceeded = currentUser.ChangePassword(model.OldPassword, ←
122                     model.NewPassword);
123             }
124             catch (Exception)
125             {
126                 changePasswordSucceeded = false;

```

```
126     }
127
128     if (changePasswordSucceeded)
129     {
130         return RedirectToAction("ChangePasswordSuccess");
131     }
132     else
133     {
134         ModelState.AddModelError("", "A senha atual ou a confirmação está incorreta.");
135     }
136 }
137
138     return View(model);
139 }
140
141 /**
142 // GET: /Usuario/ChangePasswordSuccess
143
144 public ActionResult ChangePasswordSuccess()
145 {
146     return View();
147 }
148
149
150 private IEnumerable<string> GetErrorsFromModelState()
151 {
152     return ModelState.SelectMany(x => x.Value.Errors.Select(error => error.ErrorMessage));
153 }
154
155 #region Status Codes
156 private static string ErrorCodeToString(MembershipCreateStatus createStatus)
157 {
158     // See http://go.microsoft.com/fwlink/?LinkId=177550 for
159     // a full list of status codes.
160     switch (createStatus)
161     {
162         case MembershipCreateStatus.DuplicateUserName:
163             return "Este nome de usuário já existe. Defina outro usuário.";
164
165         case MembershipCreateStatus.DuplicateEmail:
166             return "Este email já foi cadastrado. Defina outro email.";
167
168         case MembershipCreateStatus.InvalidPassword:
169             return "Senha incorreta.";
170
171         case MembershipCreateStatus.InvalidEmail:
172             return "Email inválido.";
173
174         case MembershipCreateStatus.InvalidAnswer:
175             return "Resposta inválida para recuperar a senha.";
176
177         case MembershipCreateStatus.InvalidQuestion:
178             return "Questão inválida para recuperar a senha.";
179
180         case MembershipCreateStatus.InvalidUserName:
181             return "Usuário inválido.";
182
183         case MembershipCreateStatus.ProviderError:
184             return "Ocorreu um erro durante a autenticação. Se o problema persistir, contate o administrador.";
185
186         case MembershipCreateStatus.UserRejected:
187             return "O cadastro do usuário foi cancelado. Se o problema persistir, contate o administrador.";
188
189         default:
190             return "Um erro inesperado ocorreu. Se o problema persistir, contate o administrador.";
```

```

191     }
192 }
193 #endregion
194 }
195 }
196 }
```

Código C# C.27: UsuarioController.cs

- 35** Crie uma pasta **Usuario** na pasta **Views** e acrescente os quatro arquivos abaixo.

```

1 @model K19CopaDoMundo.Models.ChangePasswordModel
2 @{
3     ViewBag.Title = "Alteração de senha";
4 }
5
6 <hgroup class="title">
7     <h1>@ViewBag.Title.</h1>
8     <h2>Utilize este formulário para alterar a sua senha.</h2>
9 </hgroup>
10
11 <p class="message-info">
12     Senhas devem ter no mínimo @Membership.MinRequiredPasswordLength caracteres.
13 </p>
14
15 @using (Html.BeginForm()) {
16     @Html.ValidationSummary()
17
18     <fieldset>
19         <legend>Alteração de Senha</legend>
20         <ol>
21             <li>
22                 @Html.LabelFor(m => m.OldPassword)
23                 @Html.PasswordFor(m => m.OldPassword)
24             </li>
25             <li>
26                 @Html.LabelFor(m => m.NewPassword)
27                 @Html.PasswordFor(m => m.NewPassword)
28             </li>
29             <li>
30                 @Html.LabelFor(m => m.ConfirmPassword)
31                 @Html.PasswordFor(m => m.ConfirmPassword)
32             </li>
33         </ol>
34         <input type="submit" value="Alterar Senha" />
35     </fieldset>
36 }
37
38 @section Scripts {
39     @Scripts.Render("~/bundles/jqueryval")
40 }
```

Código CSHTML C.10: ChangePassword.cshtml

```

1 @{
2     ViewBag.Title = "Senha alterada";
3 }
4
5 <hgroup class="title">
6     <h1>@ViewBag.Title.</h1>
7     <h2>Sua senha foi alterada com sucesso.</h2>
8 </hgroup>
```

Código CSHTML C.11: ChangePasswordSuccess.cshtml

```
1 @model K19CopaDoMundo.Models.LoginModel
2
3 @{
4     ViewBag.Title = "Log in";
5 }
6
7 <hgroup class="title">
8     <h1>@ViewBag.Title.</h1>
9     <h2>Formulário de Login</h2>
10 </hgroup>
11
12 @using (Html.BeginForm(new { ReturnUrl = ViewBag.ReturnUrl })) {
13     @Html.ValidationSummary(true, "Login não foi efetuado. Informe os dados corretos.")
14
15     <fieldset>
16         <legend>Formulário de Login</legend>
17         <ol>
18             <li>
19                 @Html.LabelFor(m => m.UserName)
20                 @Html.TextBoxFor(m => m.UserName)
21             </li>
22             <li>
23                 @Html.LabelFor(m => m.Password)
24                 @Html.PasswordFor(m => m.Password)
25             </li>
26             <li>
27                 @Html.CheckBoxFor(m => m.RememberMe)
28                 @Html.LabelFor(m => m.RememberMe, new { @class = "checkbox" })
29             </li>
30         </ol>
31         <input type="submit" value="Log in" />
32     </fieldset>
33     <p>
34         @Html.ActionLink("Registrar", "Register").
35     </p>
36 }
37
38 @section Scripts {
39     @Scripts.Render("~/bundles/jqueryval")
40 }
```

Código CSHTML C.12: Login.cshtml

```
1 @model K19CopaDoMundo.Models.RegisterModel
2
3 @{
4     ViewBag.Title = "Cadastro";
5 }
6
7 <hgroup class="title">
8     <h1>@ViewBag.Title.</h1>
9     <h2>Cadastrar</h2>
10 </hgroup>
11
12 <p class="message-info">
13     Senhas devem ter no minimo @Membership.MinRequiredPasswordLength caracteres.
14 </p>
15
16 @using (Html.BeginForm()) {
17     @Html.ValidationSummary()
18
19     <fieldset>
20         <legend>Cadastro</legend>
21         <ol>
22             <li>
23                 @Html.LabelFor(m => m.UserName)
24                 @Html.TextBoxFor(m => m.UserName)
```

```

25         </li>
26     <li>
27         @Html.LabelFor(m => m.Email)
28         @Html.TextBoxFor(m => m.Email)
29     </li>
30     <li>
31         @Html.LabelFor(m => m.Password)
32         @Html.PasswordFor(m => m.Password)
33     </li>
34     <li>
35         @Html.LabelFor(m => m.ConfirmPassword)
36         @Html.PasswordFor(m => m.ConfirmPassword)
37     </li>
38     </ol>
39     <input type="submit" value="Cadastrar" />
40   </fieldset>
41 }
42
43 @section Scripts {
44     @Scripts.Render("~/bundles/jqueryval")
45 }
```

Código CSHTML C.13: Register.cshtml

- 36** Adicione o seguinte partial View **_LoginPartial.cshtml** a pasta **Shared**.

```

1 @if (Request.IsAuthenticated) {
2     <p>
3         Olá, @Html.ActionLink(User.Identity.Name, "ChangePassword", "Usuario", ←
4             routeValues: null, htmlAttributes: new { @class = "username", title = "←
5                 Alterar senha" })!
6         @Html.ActionLink("Sair", "LogOff", "Usuario")
7     </p>
8 } else {
9     <ul>
10        <li>@Html.ActionLink("Cadastrar", "Register", "Usuario", routeValues: null, ←
11            htmlAttributes: new { id = "registerLink" })</li>
12        <li>@Html.ActionLink("Entrar", "Login", "Usuario", routeValues: null, ←
13            htmlAttributes: new { id = "loginLink" })</li>
14    </ul>
15 }
```

Código CSHTML C.14: _LoginPartial.cshtml

- 37** Altere o arquivo **_Layout.cshtml**.

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="utf-8" />
5     <title>Copa do Mundo</title>
6     <link href="/__TemplateIcon.ico" rel="shortcut icon" type="image/x-icon" />
7     <meta name="viewport" content="width=device-width" />
8     @Styles.Render("~/Content/themes/base/css", "~/Content/css")
9     @Scripts.Render("~/bundles/modernizr")
10    </head>
11    <body>
12        <header>
13            <div class="content-wrapper">
14                <div class="float-left">
15                    <p class="site-title"></p>
17                </div>
18                <div class="float-right">
```

```

18         <section id="login">
19             @Html.Partial("_LoginPartial")
20         </section>
21         <nav>
22             <ul id="menu">
23                 <li>@Html.ActionLink("Selecoes", "Index", "Selecao", null, ←
24                     new { @class = "botaو" })</li>
25                 <li> @Html.ActionLink("Jogadores", "Index", "Jogador", ←
26                     null, new { @class = "botaو" })</li>
27             </ul>
28         </nav>
29     </div>
30     </header>
31     <div id="body">
32         @RenderSection("featured", required: false)
33         <section class="content-wrapper main-content clear-fix">
34             @RenderBody()
35         </section>
36     </div>
37     <footer>
38         <div class="content-wrapper">
39             <div class="float-left">
40                 <p>&copy; @DateTime.Now.Year - K19 Copa do Mundo</p>
41             </div>
42             <div class="float-right">
43                 <ul id="social">
44                     <li><a href="http://facebook.com/k19treinamentos" class="←
45                         facebook">Facebook - K19 Treinamentos</a></li>
46                     <li><a href="http://twitter.com/k19treinamentos" class="←
47                         twitter">Twitter - K19 Treinamentos</a></li>
48                 </ul>
49             </div>
50         </footer>
51     @Scripts.Render("~/bundles/jquery")
52     @RenderSection("scripts", required: false)
53 </body>
54 </html>

```

Código CSHTML C.15: _Layout.cshtml

Adicionando um Usuário Administrador com ASP .NET Configuration

Antes de definir o filtro **Authorize** nos controladores de nosso site, vamos criar um usuário com acesso. A maneira mais fácil de criar o usuário é através do *ASP .NET Configuration*.



Exercícios de Fixação

- 38 Execute o ASP .NET Configuration que fica na aba “Solution Explorer” do Visual Studio.
- 39 Isto executará um ambiente de configuração. Abra a aba “Security” e clique no link “Enable roles”.
- 40 Posteriormente, clique em “Create or manage roles”.

41 Defina um role “Administrador” e clique *Add Role*.

42 Clique no botão “back” e crie um usuário.

43 Defina um usuário *admin* e senha *admink19!*.

Autorização Role-based

Podemos restringir o acesso as páginas com o filtro **Authorize** e podemos especificar o role que o usuário precisa ter para ter acesso a página.



Exercícios de Fixação

44 Altere o filtro de autenticação no **Web.config** para redirecionar o usuário para a action *Login* do controlador *Usuario*.

```
1 <authentication mode="Forms">
2   <forms loginUrl="~/Usuario/Login" timeout="2880" />
3 </authentication>
```

Código XML C.2: *Web.config*

45 Acrescente a seguinte string de conexão no arquivo **Web.config** para definir o local que as informações dos usuários serão armazenadas (No nosso caso, teremos duas strings de conexão).

```
1 <connectionStrings>
2   <add
3     name="K19CopaDoMundoContext" providerName="System.Data.SqlClient"
4     connectionString="Server=.\SQLEXPRESS;Database=k19copadomundo;
5     User Id=sa; Password=sa;Trusted_Connection=False;MultipleActiveResultSets=True"/>
6   <!-- Definindo o provedor para o Membership -->
7   <add
8     name="DefaultConnection" providerName="System.Data.SqlClient"
9     connectionString="Server=.\SQLEXPRESS;Database=k19copadomundo;
10    User Id=sa; Password=sa;Trusted_Connection=False;MultipleActiveResultSets=True"/>
11 </connectionStrings>
```

Código XML C.3: *Web.config*

46 Acrescente o filtro de autenticação nos controladores Selecoes e Jogadores através do atributo **Authorize**.

```
1 [Authorize(Roles = "Administrador")]
2 public class SelecaoController : Controller
```

Código C# C.28: *SelecaoController.cs*

```
1 [Authorize(Roles = "Administrador")]
2 public class JogadorController : Controller
```

Código C# C.29: JogadorController.cs

Controle de Erro

Podemos configurar uma página de erro padrão para ser utilizada toda vez que um erro ocorrer.



Exercícios de Fixação

- 47 Acrescente ao arquivo **Web.config** a tag **customErrors** para especificar a página de erro padrão. A tag **customErrors** fica dentro da tag **system.web**.

```
1 <system.web>
2 ...
3   <customErrors mode="On" defaultRedirect="~/Erro/Desconhecido">
4     <error statusCode="404" redirect="~/Erro/PaginaNaoEncontrada"/>
5   </customErrors>
6 ...
7 </system.web>
```

Código XML C.4: Web.config

- 48 Defina o controlador **Erro** e as páginas de erros padrão. As páginas de erro padrão serão criadas dentro da pasta *Views* numa subpasta *Erro*.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.Mvc;
6
7 namespace K19CopaDoMundo.Controllers
8 {
9   public class ErroController : Controller
10  {
11    //
12    // GET: /Erro/Desconhecido
13
14    public ActionResult Desconhecido()
15    {
16      return View();
17    }
18
19    //
20    // GET: /Erro/PaginaNaoEncontrada
21    public ActionResult PaginaNaoEncontrada()
22    {
23      return View();
24    }
25
26  }
27 }
```

Código C# C.30: ErroController.cs

1 @{

```

2     Layout = null;
3 }
4
5 <!DOCTYPE html>
6
7 <html>
8 <head>
9   <meta name="viewport" content="width=device-width" />
10  <title>Problema no servidor</title>
11 </head>
12 <body>
13   <h2>Desculpe, tivemos problema em nosso servidor. Volte dentro de alguns instantes<br/>.
14 </h2>
15 </body>
</html>
```

Código CSHTML C.16: Desconhecido.cshtml

```

1 @{
2     Layout = null;
3 }
4
5 <!DOCTYPE html>
6
7 <html>
8 <head>
9   <meta name="viewport" content="width=device-width" />
10  <title>Página não encontrada</title>
11 </head>
12 <body>
13   <h2>Página não encontrada</h2>
14 </body>
15 </html>
```

Código CSHTML C.17: PaginaNaoEncontrada.cshtml

Enviando email

Quando um erro ocorre na nossa aplicação, podemos permitir que o usuário envie uma email para os administradores do sistema. Para enviar as mensagens, podemos utilizar o Web



Exercícios de Fixação

- 49 Altere a tela de erro adicionando um formulário para o usuário escrever uma mensagem para os administradores da aplicação.

```

1 @{
2     Layout = null;
3 }
4
5 <!DOCTYPE html>
6
7 <html>
8 <head>
9   <meta name="viewport" content="width=device-width" />
10  <title>Problema no servidor</title>
11 </head>
12 <body>
13   <h2>Desculpe, tivemos problema em nosso servidor. Volte dentro de alguns instantes<br/>.
14 </h2>
15 </body>
</html>
```

```

14 <p>Envie uma mensagem para os administradores do sistema.</p>
15     @using (Html.BeginForm("Envia", "Email"))
16     {
17         <div class="editor-label">
18             @Html.Label("Mensagem")
19         </div>
20         <div class="editor-field">
21             @Html.TextArea("Mensagem")
22         </div>
23         <input type="submit" value="Enviar" />
24     }
25 </body>
26 </html>

```

Código CSHTML C.18: Desconhecido.cshtml

- 50** Crie um controlador que envie as mensagens por email utilizando o helper **WebMail**. Observação, utilize usuários, senhas e emails válidos do gmail para este exercício.

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.Mvc;
6 using System.Web.Helpers;
7
8 namespace K19CopaDoMundo.Controllers
9 {
10     public class EmailController : Controller
11     {
12
13         public EmailController()
14         {
15             WebMail.SmtpServer = "smtp.gmail.com";
16             WebMail.EnableSsl = true;
17             WebMail.SmtpPort = 587;
18             WebMail.From = "USUARIO@gmail.com";
19             WebMail.UserName = "USUARIO@gmail.com";
20             WebMail.Password = "SENHA";
21         }
22
23         // POST: /Email/Envia
24         [HttpPost]
25         public ActionResult Envia(string mensagem)
26         {
27
28             WebMail.Send("EMAIL", "Copa do Mundo - Erro", mensagem);
29             return View();
30         }
31
32     }
33 }

```

Código C# C.31: EmailController.cs

- 51** Crie uma página **Envia.cshtml** para mostrar ao usuário que a mensagem foi enviada com sucesso e acrescente um link para a página inicial do site.

```

1 @{
2     Layout = null;
3 }
4
5 <!DOCTYPE html>

```

```
6 <html>
7 <head>
8     <title>Envia</title>
9 </head>
10 <body>
11     <div>
12         Mensagem enviada com sucesso.
13     </div>
14     <div>
15         @Html.ActionLink("Voltar para página inicial", "Index", "Selecoes")
16     </div>
17 </body>
18 </html>
```

Código CSHTML C.19: *Envia.cshtml*





RESPOSTAS

Resposta do Exercise 2.1

```

1  using System.Data.Odbc;
2
3  namespace Odbc
4  {
5      class InsereLivro
6      {
7          static void Main(string[] args)
8          {
9              string stringDeConexao = @"driver={SQL Server};
10                 server=MARCELO-PC\SQLEXPRESS;database=livraria;uid=sa;pwd=sa;";
11
12             System.Console.Write("Digite o Título do Livro:");
13             string titulo = System.Console.ReadLine();
14
15             System.Console.Write("Digite o Preço do Livro:");
16             string preco = System.Console.ReadLine();
17
18             System.Console.Write("Digite o Id da Editora do Livro:");
19             string editoraId = System.Console.ReadLine();
20
21             string textoInsereEditora =
22                 @"INSERT INTO Livros (Titulo, Preco, EditoraId)
23                 VALUES('" + titulo + @"', '" + preco + @"', " + editoraId + ")";
24
25             using (OdbcConnection conexao = new OdbcConnection(stringDeConexao))
26             {
27                 OdbcCommand command = new OdbcCommand(textoInsereEditora, conexao);
28                 conexao.Open();
29                 command.ExecuteNonQuery();
30             }
31         }
32     }
33 }
```

Código C# 2.9: *InsereLivro.cs*

Resposta do Exercise 2.3

```

1  using System.Data.Odbc;
2
3  namespace Odbc
4  {
5      class InsereLivro
6      {
7          static void Main(string[] args)
8          {
9              string stringDeConexao = @"driver={SQL Server};
10                 server=MARCELO-PC\SQLEXPRESS;database=livraria;uid=sa;pwd=sa;";
```

```

11     System.Console.Write("Digite o Título do Livro:");
12     string titulo = System.Console.ReadLine();
13
14     System.Console.Write("Digite o Preço do Livro:");
15     string preco = System.Console.ReadLine();
16
17     System.Console.Write("Digite o Id da Editora do Livro:");
18     string editoraId = System.Console.ReadLine();
19
20     string textoInsereEditora =
21         @"INSERT INTO Livros (Titulo, Preco, EditoraId) VALUES (?,?,?,?)";
22
23     using (OdbcConnection conexao = new OdbcConnection(stringDeConexao))
24     {
25         OdbcCommand command = new OdbcCommand(textoInsereEditora, conexao);
26
27         command.Parameters.AddWithValue("@Titulo", titulo);
28         command.Parameters.AddWithValue("@Preco", preco);
29         command.Parameters.AddWithValue("@EditoraId", editoraId);
30
31         conexao.Open();
32         command.ExecuteNonQuery();
33     }
34 }
35 }
36 }
37 }
```

Código C# 2.14: InsereLivro.cs

Resposta do Exercise 2.5

```

1  using System.Data.Odbc;
2
3 namespace Odbc
4 {
5     class ListaLivro
6     {
7         static void Main(string[] args)
8     {
9         string stringDeConexao = @"driver={SQL Server};
10         server=MARCELO-PC\SQLEXPRESS;database=livraria;uid=sa;pwd=sa;";
11
12         using (OdbcConnection conexao = new OdbcConnection(stringDeConexao))
13     {
14             string textoListaEditora = "SELECT * FROM Livros";
15             OdbcCommand command = new OdbcCommand(textoListaEditora, conexao);
16             conexao.Open();
17             OdbcDataReader resultado = command.ExecuteReader();
18
19             while (resultado.Read())
20             {
21                 long? id = resultado["Id"] as long?;
22                 string titulo = resultado["Titulo"] as string;
23                 double? preco = resultado["Preco"] as double?;
24                 long? editoraId = resultado["EditoraId"] as long?;
25
26                 System.Console.WriteLine("{0} : {1} - {2} - {3}\n",
27                     id, titulo, preco, editoraId);
28             }
29         }
30     }
31 }
32 }
```

Código C# 2.21: *ListaLivro.cs*

Resposta do Exercise 2.6

```

1  using System.Data.Odbc;
2
3  namespace Odbc
4  {
5      class InsereLivro
6      {
7          static void Main(string[] args)
8          {
9              System.Console.Write("Digite o Título do Livro:");
10             string titulo = System.Console.ReadLine();
11
12             System.Console.Write("Digite o Preço do Livro:");
13             string preco = System.Console.ReadLine();
14
15             System.Console.Write("Digite o Id da Editora do Livro:");
16             string editoraId = System.Console.ReadLine();
17
18             string textoInsereEditora =
19                 @"INSERT INTO Livros (Titulo, Preco, EditoraId) VALUES (?,?,?)";
20
21             using (OdbcConnection conexao = ConnectionFactory.CreateConnection())
22             {
23                 OdbcCommand command = new OdbcCommand(textoInsereEditora, conexao);
24
25                 command.Parameters.AddWithValue("@Titulo", titulo);
26                 command.Parameters.AddWithValue("@Preco", preco);
27                 command.Parameters.AddWithValue("@EditoraId", editoraId);
28
29                 conexao.Open();
30                 command.ExecuteNonQuery();
31             }
32         }
33     }
34 }
```

Código C# 2.26: *InsereLivro.cs*

```

1  using System.Data.Odbc;
2
3  namespace Odbc
4  {
5      class ListaLivro
6      {
7          static void Main(string[] args)
8          {
9              using (OdbcConnection conexao = ConnectionFactory.CreateConnection())
10             {
11                 string textoListaEditora = "SELECT * FROM Livros";
12                 OdbcCommand command = new OdbcCommand(textoListaEditora, conexao);
13                 conexao.Open();
14                 OdbcDataReader resultado = command.ExecuteReader();
15
16                 while (resultado.Read())
17                 {
18                     long? id = resultado["Id"] as long?;
19                     string titulo = resultado["Titulo"] as string;
20                     double? preco = resultado["Preco"] as double?;
```

```
21     long? editoraId = resultado["EditoraId"] as long?;
22
23     System.Console.WriteLine("{0} : {1} - {2} - {3}\n",
24         id, titulo, preco, editoraId);
25     }
26   }
27 }
28 }
```

Código C# 2.27: *ListaLivro.cs*

Resposta do Exercise 5.1

```
1 @{
2   ViewBag.Title = "Index";
3 }
4 @for(int i = 0; i < 10; i++)
5 {
6   <h2>Olá @i</h2>
7 }
```

Código CSHTML 5.10: *Index.cshtml*