



K9

TREINAMENTOS

Desenvolvimento Web com Struts

Desenvolvimento Web com Struts2 e JPA2

14 de junho de 2015

As apostilas atualizadas estão disponíveis em www.k19.com.br

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 MySQL Server	2
1.3 Bases de dados (<i>Databases</i>)	2
1.4 Criando uma base de dados no MySQL Server	2
1.5 Tabelas	3
1.6 Criando tabelas no MySQL Server	3
1.7 Operações Básicas	4
1.8 Chaves Primária e Estrangeira	5
1.9 Exercícios de Fixação	5
1.10 Exercícios Complementares	9
2 JDBC	27
2.1 Driver	28
2.2 JDBC	29
2.3 Instalando o Driver JDBC do MySQL Server	30
2.4 Criando uma conexão	30
2.5 Inserindo registros	31
2.6 Exercícios de Fixação	31
2.7 Exercícios Complementares	33
2.8 SQL Injection	33
2.9 Exercícios de Fixação	34
2.10 Exercícios Complementares	35

2.11	Listando registros	35
2.12	Exercícios de Fixação	36
2.13	Exercícios Complementares	37
2.14	Connection Factory	37
2.15	Exercícios de Fixação	38
2.16	Exercícios Complementares	40
2.17	Desafios	40
3	JPA 2 e Hibernate	41
3.1	Múltiplas sintaxes da linguagem SQL	41
3.2	Orientação a Objetos VS Modelo Relacional	41
3.3	Ferramentas ORM	42
3.4	O que é JPA e Hibernate?	43
3.5	Bibliotecas	43
3.6	Configuração	44
3.7	Mapeamento	44
3.8	Gerando Tabelas	46
3.9	Exercícios de Fixação	46
3.10	Manipulando entidades	48
3.11	Exercícios de Fixação	50
3.12	Repository	51
3.13	Exercícios de Fixação	52
4	Web Container	55
4.1	Necessidades de uma aplicação web	55
4.2	Web Container	56
4.3	Servlet e Java EE	57
4.4	Exercícios de Fixação	57
4.5	Aplicação Web Java	58
4.6	Exercícios de Fixação	59
4.7	Processando requisições	59
4.8	Servlet	59
4.9	Exercícios de Fixação	60
4.10	Frameworks	61
5	Visão Geral do Struts 2	63
5.1	MVC e Front Controller	63
5.2	Bibliotecas	63
5.3	Configurando uma aplicação Struts	64
5.4	Logging	65
5.5	Actions	65
5.6	Exemplo Prático	69
5.7	Exercícios de Fixação	71
6	Integração Struts e JPA	77
6.1	Bibliotecas	77
6.2	Configuração	77
6.3	Mapeamento	78
6.4	Inicialização e Finalização	78
6.5	Transações	80

6.6 Recuperando o EntityManager da Requisição	81
6.7 Exercícios de Fixação	81
7 Autenticação	87
7.1 Exercícios de Fixação	87
8 Páginas de Erro	93
8.1 Exercícios de Fixação	93
9 CRUD	97
9.1 Exercícios de Fixação	97
A Projeto	103
A.1 Exercícios de Fixação	103





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 tornam-se 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

- SQL Server
- MySQL Server
- PostgreSQL



MySQL Server

Neste treinamento, utilizaremos o MySQL Server, que é mantido pela Oracle e amplamente utilizado em aplicações comerciais. Para instalar o MySQL Server, você pode utilizar o artigo disponível em nosso site: <http://www.k19.com.br/artigos/installando-mysql/>



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 MySQL Server

Para criar uma base de dados no MySQL Server, podemos utilizar o comando **CREATE DATABASE**.

```
mysql> CREATE DATABASE livraria;
Query OK, 1 row affected (0.02 sec)
```

Terminal 1.1: Criando uma base de dados.

Podemos utilizar o comando **SHOW DATABASES** para listar as bases de dados existentes.

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| livraria      |
| mysql          |
| test           |
+-----+
4 rows in set (0.03 sec)
```

Terminal 1.2: Listando as bases de dados existentes.

Repare que, além da base de dados **livraria**, há outras três bases. Essas bases foram criadas automaticamente pelo próprio MySQL Server para teste ou para armazenar configurações.

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

```
mysql> DROP DATABASE livraria;
Query OK, 0 rows affected (0.08 sec)
```

Terminal 1.3: Destruindo uma base de dados.



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.

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

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

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

```
mysql> CREATE TABLE 'livraria'.'Livro' (
-> 'titulo' VARCHAR(255),
-> 'preco' DOUBLE
-> )
-> ENGINE=MyISAM;
Query OK, 0 rows affected (0.14 sec)
```

Terminal 1.4: Criando uma tabela.

As tabelas de uma base de dados podem ser listadas através do comando **SHOW TABLES**. Antes de utilizar esse comando, devemos selecionar uma base de dados através do comando **USE**.

```
mysql> USE livraria;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> SHOW TABLES;
```

```
+-----+
| Tables_in_livraria |
+-----+
| Livro             |
+-----+
1 row in set (0.00 sec)
```

Terminal 1.5: Listando as tabelas de uma base de dados.

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

```
mysql> DROP TABLE Livro;
Query OK, 0 rows affected (0.00 sec)
```

Terminal 1.6: Destruindo uma tabela.



Operações Básicas

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.

```
mysql> INSERT INTO Livro (titulo, preco) VALUES ('Java', 98.75);
Query OK, 1 row affected (0.00 sec)
```

Terminal 1.7: Inserindo um registro.

```
+-----+
| titulo | preco |
+-----+
| Java   | 98.75 |
+-----+
1 row in set (0.00 sec)
```

Terminal 1.8: Selecionando registros.

```
mysql> UPDATE Livro SET preco = 115.9 WHERE titulo = 'Java';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1    Changed: 1    Warnings: 0
```

Terminal 1.9: Alterando registros.

```
+-----+
| titulo | preco |
+-----+
| Java   | 115.9 |
+-----+
1 row in set (0.00 sec)
```

Terminal 1.10: Selecionando registros.

```
mysql> DELETE FROM Livro WHERE titulo = 'Java';
Query OK, 1 row affected (0.00 sec)
```

Terminal 1.11: Removendo registros.

```
mysql> SELECT * FROM Livro;
Empty set (0.00 sec)
```

Terminal 1.12: Selecionando registros.



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 MySQL Server, 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 um terminal, crie e acesse uma pasta com o seu nome.

```
cosen@k19:~$ mkdir rafael
cosen@k19:~$ cd rafael/
cosen@k19:~/rafael$
```

Terminal 1.13: Criando e acessando uma pasta com o seu nome.

- 2 Estando dentro da sua pasta, acesse o **MySQL Server** utilizando o usuário **root** e a senha **root**.

```
k19@k19-11:~/rafael$ mysql -u root -p
Enter password:
```

Terminal 1.14: Logando no MySQL Server.

- 3 Caso exista uma base de dados chamada **livraria**, remova-a. Utilize o comando **SHOW DATABASES** para listar as bases de dados existentes e o comando **DROP DATABASE** para remover a base **livraria** se ela existir.

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| livraria |
| mysql |
| test |
+-----+
4 rows in set (0.00 sec)

mysql> DROP DATABASE livraria;
Query OK, 1 row affected (0.12 sec)
```

Terminal 1.15: Listando as bases de dados existentes e removendo a base livraria.

- 4 Crie uma nova base de dados chamada **livraria**. Utilize o comando **CREATE DATABASE**. Você vai utilizar esta base nos exercícios seguintes.

```
mysql> CREATE DATABASE livraria;
Query OK, 1 row affected (0.00 sec)
```

Terminal 1.16: Criando a base livraria.

- 5 Abra um editor de texto e digite o código abaixo para criar uma tabela com o nome **Editora**. Depois salve o arquivo com o nome **create-table-editora.sql** dentro da pasta com o seu nome.

```
1 USE livraria;
2 CREATE TABLE Editora (
3     id BIGINT NOT NULL AUTO_INCREMENT ,
4     nome VARCHAR (255) NOT NULL ,
5     email VARCHAR (255) NOT NULL ,
6     PRIMARY KEY (id)
7 )
8 ENGINE = InnoDB;
```

Código SQL 1.1: Criando a tabela Editora

- 6 Dentro do terminal, use o comando source para executar o arquivo que você acabou de criar.

```
mysql> source create-table-editora.sql
Database changed
Query OK, 0 rows affected (0.08 sec)
```

Terminal 1.17: Executando a tabela Editora.

- 7 Abra um novo editor de texto e digite o código abaixo para criar uma tabela com o nome **Livro**. Em seguida, salve o arquivo com o nome `create-table-livro.sql` dentro da pasta com o seu nome.

```
1 USE livraria;
2 CREATE TABLE Livro (
3     id BIGINT NOT NULL AUTO_INCREMENT,
4     titulo VARCHAR(255) NOT NULL,
5     preco DOUBLE NOT NULL,
6     editora_id BIGINT NOT NULL,
7     PRIMARY KEY(id),
8     CONSTRAINT fk_editora FOREIGN KEY fk_editora(editora_id)
9     REFERENCES Editora(id)
10    ON DELETE RESTRICT
11    ON UPDATE RESTRICT
12 )
13 ENGINE = InnoDB;
```

Código SQL 1.2: Criando a tabela Livro

- 8 Dentro do terminal, use o comando source para executar o código do arquivo `create-table-livro.sql`.

```
mysql> source create-table-livro.sql
Database changed
Query OK, 0 rows affected (0.08 sec)
```

Terminal 1.18: Executando a tabela Livro.

- 9 Abra um novo editor de texto e digite o código abaixo para adicionar alguns registros na tabela **Editora**. Depois salve o arquivo com o nome `adicionando-registros-editora.sql` dentro da pasta com o seu nome.

```
1 INSERT INTO Editora (nome, email) VALUES ('Oreilly', 'oreilly@email.com');
2
3 INSERT INTO Editora (nome, email) VALUES ('Wrox', 'wrox@email.com');
4
5 INSERT INTO Editora (nome, email) VALUES ('Apress', 'apress@email.com');
```

Código SQL 1.3: Adicionando registros na tabela Editora

- 10 Dentro do terminal, execute o arquivo que você acabou de criar para adicionar alguns registro na tabela **Editora**.

```
mysql> source adicionando-registros-editora.sql
Query OK, 1 row affected (0.03 sec)

Query OK, 1 row affected (0.04 sec)

Query OK, 1 row affected (0.04 sec)
```

Terminal 1.19: Inserindo editoras.

- 11** Abra um novo editor de texto e digite o código abaixo para adicionar alguns registros na tabela **Livro**. Depois salve o arquivo com o nome adicionando-registros-livro.sql dentro da pasta com o seu nome.

```

1 INSERT INTO Livro (titulo, preco, editora_id) VALUES ('Aprendendo C#', 89.90, 1);
2
3 INSERT INTO Livro (titulo, preco, editora_id) VALUES ('Introdução ao JSF 2',
4 122.90, 3);
5
6 INSERT INTO Livro (titulo, preco, editora_id) VALUES ('JSF 2 Avançado', 149.90, 3);

```

Código SQL 1.4: Adicionando alguns registros na tabela Livro

- 12** Dentro do terminal, execute o arquivo que você acabou de criar para adicionar alguns registros na **Livro**.

```

mysql> source adicionando-registros-livro.sql
Query OK, 1 row affected (0.02 sec)

Query OK, 1 row affected (0.04 sec)

Query OK, 1 row affected (0.04 sec)

```

Terminal 1.20: Inserindo livros.

- 13** Consulte os registros da tabela Editora e da tabela Livro. Utilize o comando **SELECT**.

```

mysql> SELECT * FROM Editora;
+----+-----+-----+
| id | nome | email |
+----+-----+-----+
| 1 | Oreilly | oreilly@email.com |
| 2 | Wrox | wrox@email.com |
| 3 | Apress | apress@email.com |
+----+-----+-----+
3 rows in set (0.00 sec)

```

Terminal 1.21: Selecionando as editoras.

```

mysql> SELECT * FROM Livro;
+----+-----+-----+-----+
| id | titulo | preco | editora_id |
+----+-----+-----+-----+
| 1 | Aprendendo C# | 89.9 | 1 |
| 2 | Introdução ao JSF 2 | 122.9 | 3 |
| 3 | JSF 2 Avançado | 149.9 | 3 |
+----+-----+-----+-----+
3 rows in set (0.00 sec)

```

Terminal 1.22: Selecionando os livros.

- 14** Altere alguns dos registros da tabela Livro. Utilize o comando **UPDATE**.

```

mysql> UPDATE Livro SET preco=92.9 WHERE id=1;
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0

```

Terminal 1.23: Alterando livros.

- 15** Altere alguns dos registros da tabela Editora. Utilize o comando **UPDATE**.

```
mysql> UPDATE Editora SET nome='OReilly' WHERE id=1;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Terminal 1.24: Alterando editoras.

- 16 Remova alguns registros da tabela Livro. Utilize o comando **DELETE**.

```
mysql> DELETE FROM Livro WHERE id=2;
Query OK, 1 row affected (0.07 sec)
```

Terminal 1.25: Removendo livros.

- 17 Remova alguns registros da tabela Editora. Preste atenção para não remover uma editora que tenha algum livro relacionado já adicionado no banco. Utilize o comando **DELETE**.

```
mysql> DELETE FROM Editora WHERE id=2;
Query OK, 1 row affected (0.05 sec)
```

Terminal 1.26: Removendo editoras.

- 18 Faça uma consulta para buscar todos os livros de uma determinada editora.

```
mysql> SELECT * FROM Livro as L, Editora as E WHERE L.editora_id=E.id and E.id = 1;
+----+-----+-----+-----+-----+
| id | titulo | preco | editora_id | id | nome   | email
+----+-----+-----+-----+-----+
| 1  | Aprendendo C# | 92.9 | 1 | 1 | OReilly | oreilly@email.com |
+----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

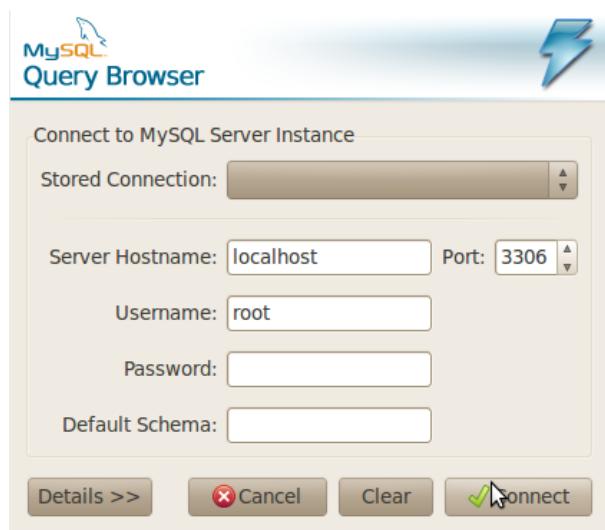
Terminal 1.27: Selecionando os livros de uma editora.



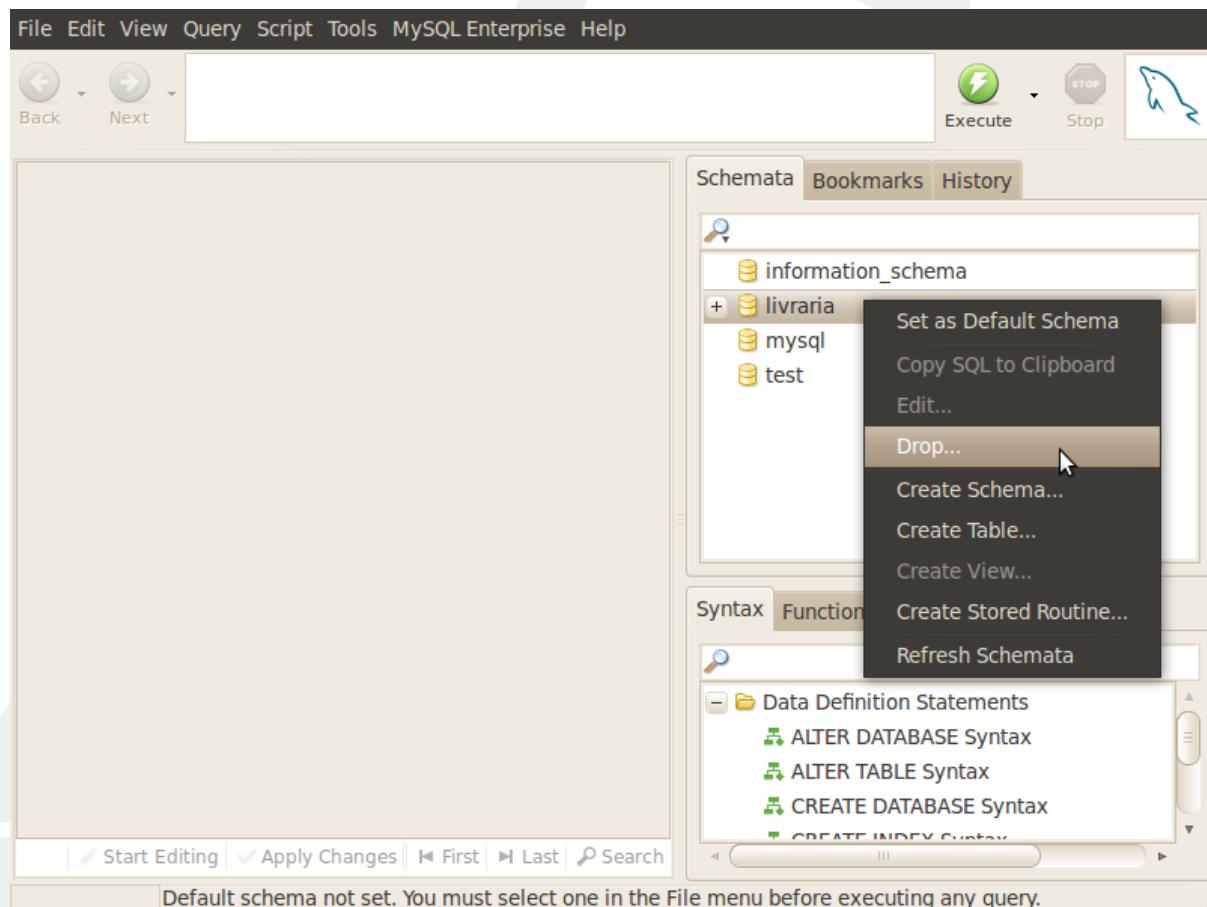
Exercícios Complementares

Utilize o MySQL Query Browser para refazer os exercícios anteriores.

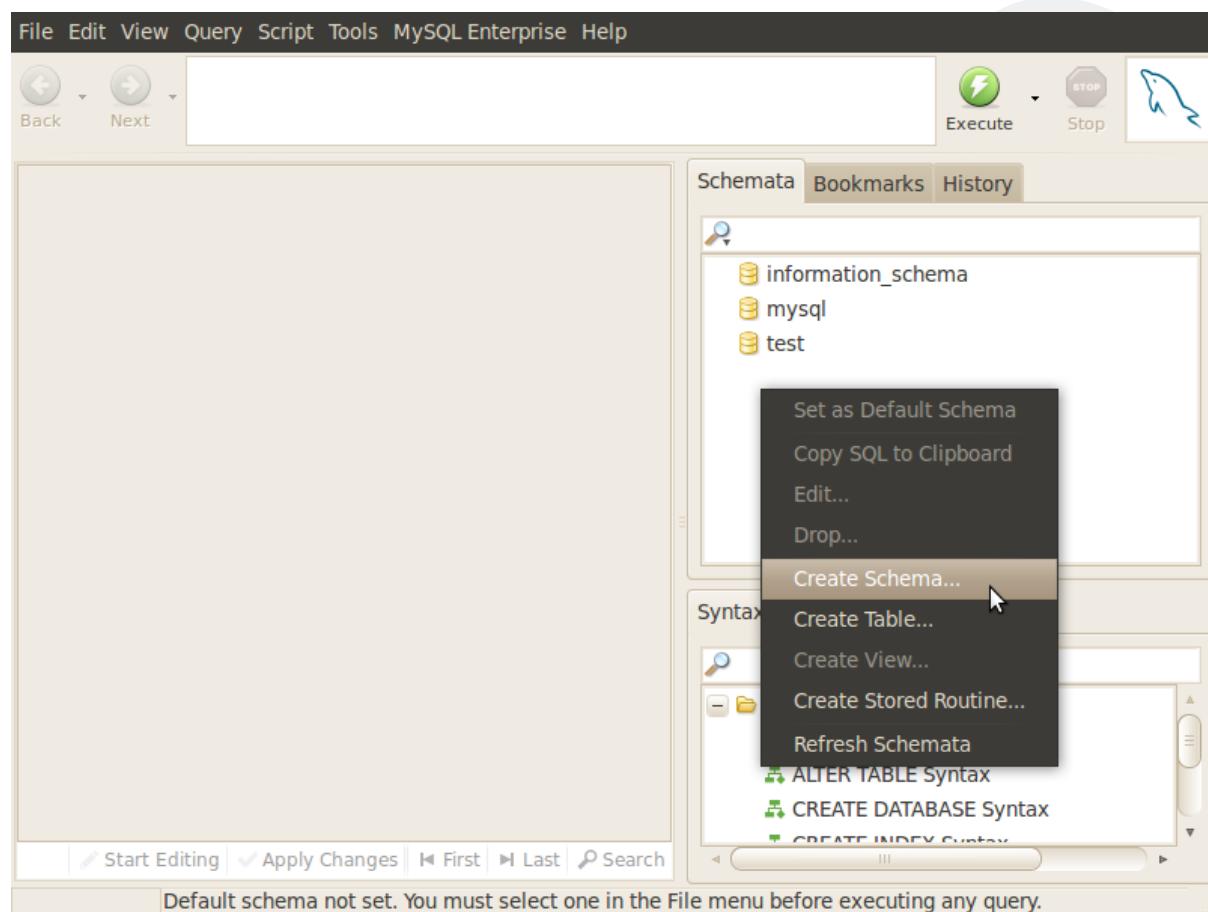
- 1 Abra o **MySQL Query Browser** utilizando **localhost** como Server Hostname, **root** como User-name e **root** como Password.



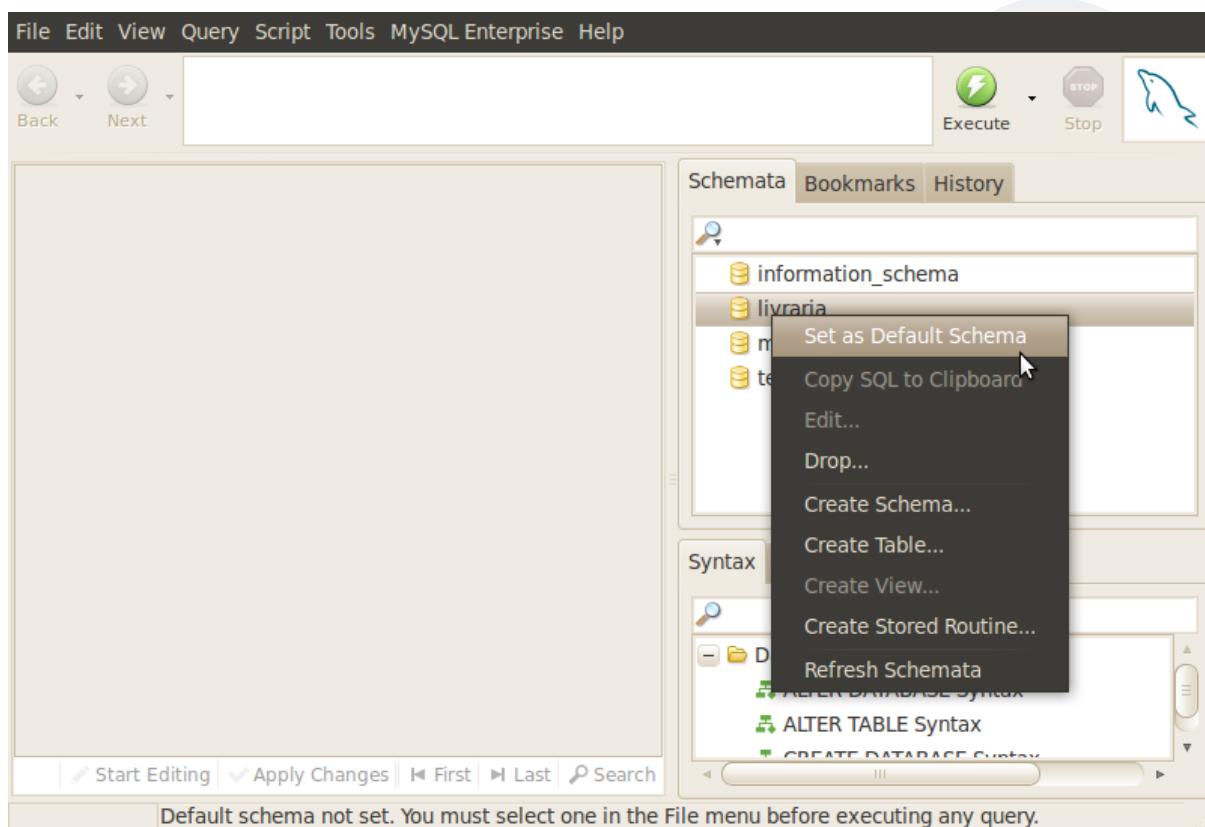
- 2 Caso exista uma base de dados chamada **livraria**, remova-a conforme a figura abaixo.



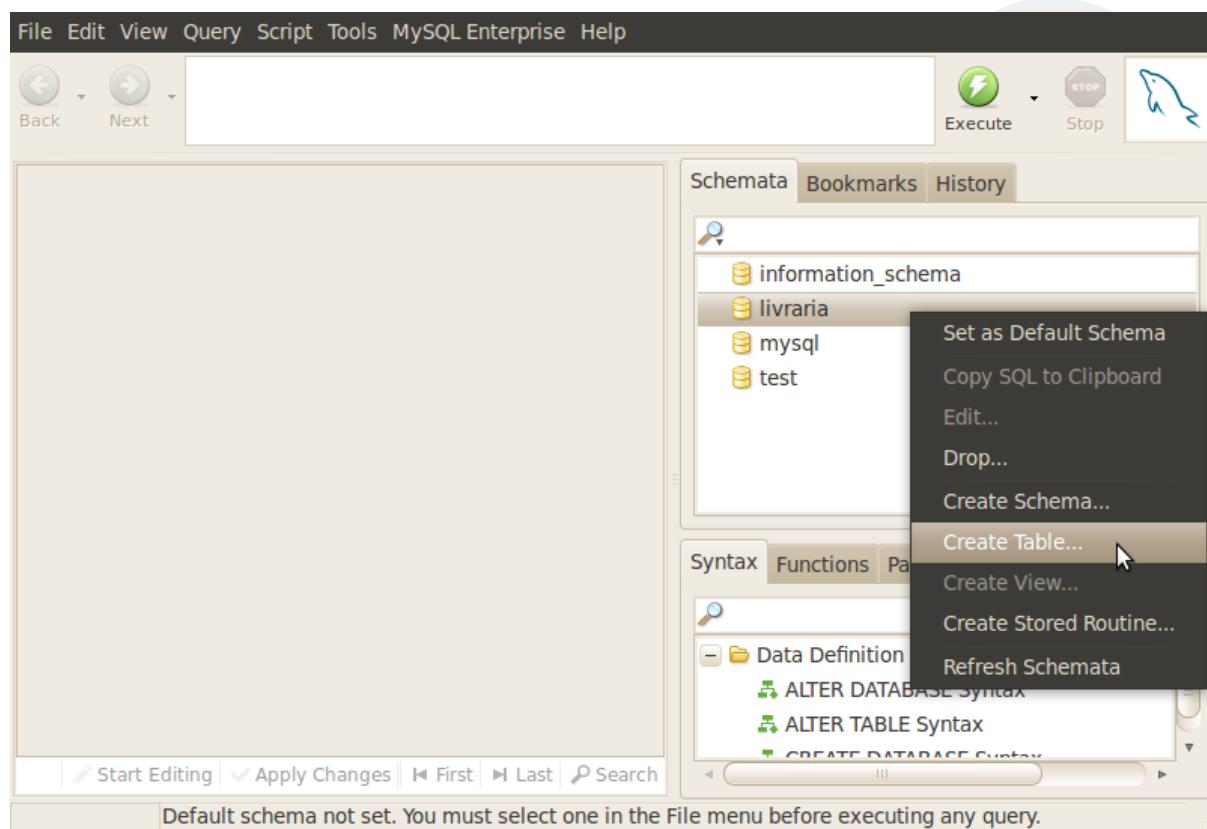
- 3 Crie uma nova base de dados chamada **livraria**, conforme mostrado na figura abaixo. Você vai utilizar esta base nos exercícios seguintes.



- 4 Seleione a base de dados livraria como padrão.



- 5 Crie uma tabela chamada **Editora** conforme as figuras abaixo.



Altere o modo de criação da tabela para **InnoDB**, conforme mostrado na figura.

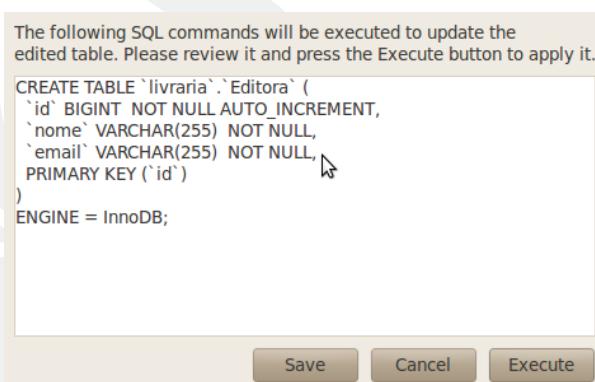


Crie os campos conforme a figura e não esqueça de tornar todos os campos obrigatórios, marcando a opção **NOT NULL**. Além disso, o campo **id** deve ser uma chave primária e automaticamente incrementada.

Column Name	Data Type	NOT NULL	AUTO INC	Flags	Default Value	Comments
id	BIGINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
nome	VARCHAR(255)	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
email	VARCHAR(255)	<input checked="" type="checkbox"/>	<input type="checkbox"/>			

Name: id	Data Type: BIGINT	Default Value: <input type="text"/> NULL
Column Options	Flags:	Character Set: <input type="button"/>
<input checked="" type="checkbox"/> Primary Key	<input type="checkbox"/> UNSIGNED	Collation: <input type="button"/>
<input checked="" type="checkbox"/> Not NULL	<input type="checkbox"/> ZEROFILL	
<input checked="" type="checkbox"/> Auto Increment		

Após clicar no botão “Apply Changes”, aparecerá uma janela mostrando os comandos SQL gerados. Clique no botão “Execute”.



- 6 Crie uma tabela chamada **Livro** conforme as figuras abaixo. Altere o modo de criação da tabela para **InnoDB**, conforme mostrado na figura.

Table Name: Livro Comment:

Columns and Indices Table Options Advanced Options

Storage Engine: InnoDB

Supports transactions, row-level locking, and foreign keys

Default Character Set

Character Set: The default character set that is used for the table. Starting from MySQL 4.1 you can specify an individual character set for each column.

Collation: Collation method that is used to compare text and sort columns.

Novamente, adicione os campos conforme a figura abaixo, lembrando de marcar a opção **NOT NULL**. Além disso, o campo **id** deve ser uma chave primária e automaticamente incrementada.

Table Name: Livro Comment:

Columns and Indices Table Options Advanced Options

Column Name	Data Type	NOT NULL	AUTO INC	Flags	Default Value	Comments
id	BIGINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
titulo	VARCHAR(255)	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
preco	DOUBLE	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
editora_id	BIGINT	<input checked="" type="checkbox"/>	<input type="checkbox"/>			

Column Details Indices Foreign Keys

Name: id Data Type: BIGINT Default Value: NULL

Column Options Flags: UNSIGNED ZEROFILL Character Set:

Primary Key Not NULL Auto Increment Collation:

Comment:

Você precisa tornar o campo **editora_id** uma chave estrangeira. Selecione a aba **Foreign Keys** e clique no botão com o símbolo “+” para adicionar uma chave estrangeira. Depois, siga os procedimentos conforme mostrados na figura abaixo.

Table Name: Livro Comment:

Columns and Indices Table Options Advanced Options

Column Name	Data Type	NOT NULL	AUTO INC	Flags	Default Value	Comments
id	BIGINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
titulo	VARCHAR(255)	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
preco	DOUBLE	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
editora_id	BIGINT	<input checked="" type="checkbox"/>	<input type="checkbox"/>			

Column Details Indices Foreign Keys

Foreign Key Settings

Key Name: fk_editora
On Delete: RESTRICT
On Update: RESTRICT
Refer. Table: Editora

Column	Foreign Column
editora_id	id

Use drag and drop to add columns to the list above.

Discard Changes Apply Changes Close

The following SQL commands will be executed to update the edited table. Please review it and press the Execute button to apply it.

```
CREATE TABLE `livraria`.`Livro` (
  `id` BIGINT NOT NULL AUTO_INCREMENT,
  `titulo` VARCHAR(255) NOT NULL,
  `preco` DOUBLE NOT NULL,
  `editora_id` BIGINT NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_editora` FOREIGN KEY `fk_editora`(`editora_id`)
    REFERENCES `Editora`(`id`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT
)
ENGINE = InnoDB;
```

Save Cancel Execute

- 7 Adicione alguns registros na tabela Editora. Veja exemplos na figura abaixo.

The screenshot shows the MySQL Enterprise Workbench interface. In the top menu bar, the following items are visible: File, Edit, View, Query, Script, Tools, MySQL Enterprise, and Help. Below the menu bar are standard toolbar icons for Undo, Redo, Open, Save, Continue, Step, Execute, and Stop. A status bar at the bottom left shows the time as 11:26.

The main area contains two tabs: "Resultset 1" and "New Script". The "Resultset 1" tab is active and displays the following SQL code:

```
1 • INSERT INTO Editora (nome, email)
2   VALUES ('Oreilly', 'oreilly@email.com');
3
4 • INSERT INTO Editora (nome, email)
5   VALUES ('Wrox', 'wrox@email.com');
6
7 INSERT INTO Editora (nome, email)
8   VALUES ('Apress', 'apress@email.com');
```

To the right of the code editor is a "Schemata" browser window. It lists the databases: information_schema, livraria, mysql, and test. The "livraria" database is expanded, showing its tables: Editora and Livro.

Below the schemata browser is a "Syntax" panel with tabs for Functions, Params, and Trx. The Functions tab is selected, showing a list of database statements: ALTER DATABASE Syntax, ALTER TABLE Syntax, and CREATE DATABASE Syntax.

- 8 Adicione alguns registros na tabela Livro. Veja exemplos na figura abaixo.

The screenshot shows the MySQL Enterprise Workbench interface. In the top menu bar, the options File, Edit, View, Query, Script, Tools, MySQL Enterprise, and Help are visible. Below the menu is a toolbar with Undo, Redo, Open, Save, Continue, Step, Execute, and Stop buttons. A status bar at the bottom shows the time as 11:36.

The main area contains two tabs: Resultset 1 and New Script. The Resultset 1 tab displays the following SQL script:

```
1 • INSERT INTO Livro (titulo, preco, editora_id)
2     VALUES ('Aprendendo C#', 89.90, 1);
3
4 • INSERT INTO Livro (titulo, preco, editora_id)
5     VALUES ('Introdução ao JSF 2', 122.90, 3);
6
7 • INSERT INTO Livro (titulo, preco, editora_id)
8     VALUES ('JSF 2 Avançado', 149.90, 3);
```

To the right of the script editor is a database browser window titled "Schema". It shows the "livraria" database selected, with its tables "Editora" and "Livro" listed. Other databases like "information_schema", "mysql", and "test" are also visible. Below the schema browser is a "Syntax" panel containing links to Data Definition Statements, ALTER DATABASE Syntax, ALTER TABLE Syntax, and CREATE DATABASE Syntax.

- 9 Consulte os registros da tabela Editora e, em seguida, consulte a tabela Livro. Veja exemplos logo abaixo.

File Edit View Query Script Tools MySQL Enterprise Help

`SELECT * FROM Editora;`

Back Next Execute Stop

Resultset 1 New Script

	id	nome	email
1	Oreilly	oreilly@email.com	
2	Wrox	wrox@email.com	
3	Apress	apress@email.com	

3 rows fetched in 0:00.0240 Start Editing Apply Changes First Last Search

Query finished.

Schemata Bookmarks History

- information_schema
- livraria**
 - Editora
 - Livro
- mysql
- test

Syntax Functions Params Trx

- Data Definition Statement
 - ALTER DATABASE Syntax
 - ALTER TABLE Syntax
 - CREATE DATABASE Syntax

File Edit View Query Script Tools MySQL Enterprise Help

`SELECT * FROM Livro;`

Back Next Execute Stop

Resultset 1 New Script

	id	titulo	preco	editora_id
1	Aprendendo C#	89.9	1	
2	Introdução ao JSF 2	122.9	3	
3	JSF 2 Avançado	149.9	3	

3 rows fetched in 0:00.0518 Start Editing Apply Changes First Last Search

Query finished.

Schemata Bookmarks History

- information_schema
- livraria**
 - Editora
 - Livro
- mysql
- test

Syntax Functions Params Trx

- Data Definition Statement
 - ALTER DATABASE Syntax
 - ALTER TABLE Syntax
 - CREATE DATABASE Syntax

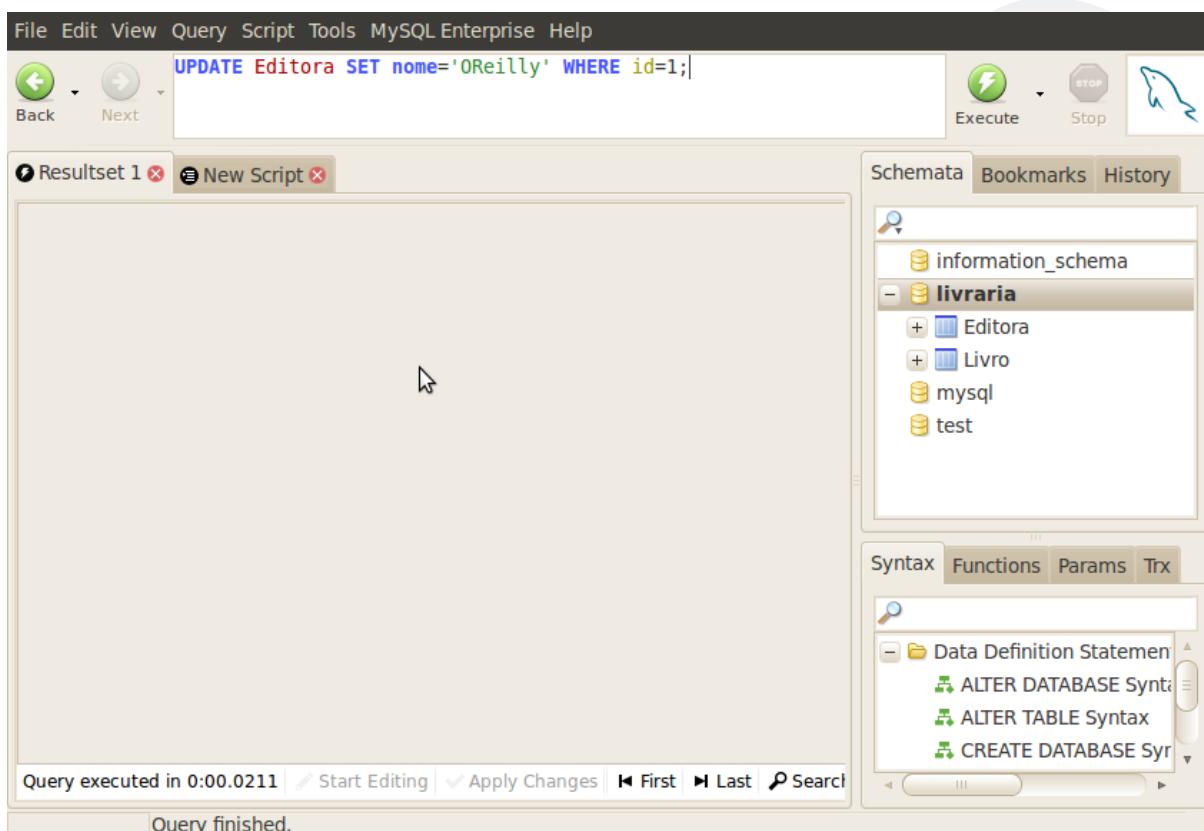
- 10 Altere alguns dos registros da tabela Livro. Veja o exemplo abaixo.

The screenshot shows the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Script, Tools, MySQL Enterprise, and Help. The main area contains a query editor with the following SQL command:

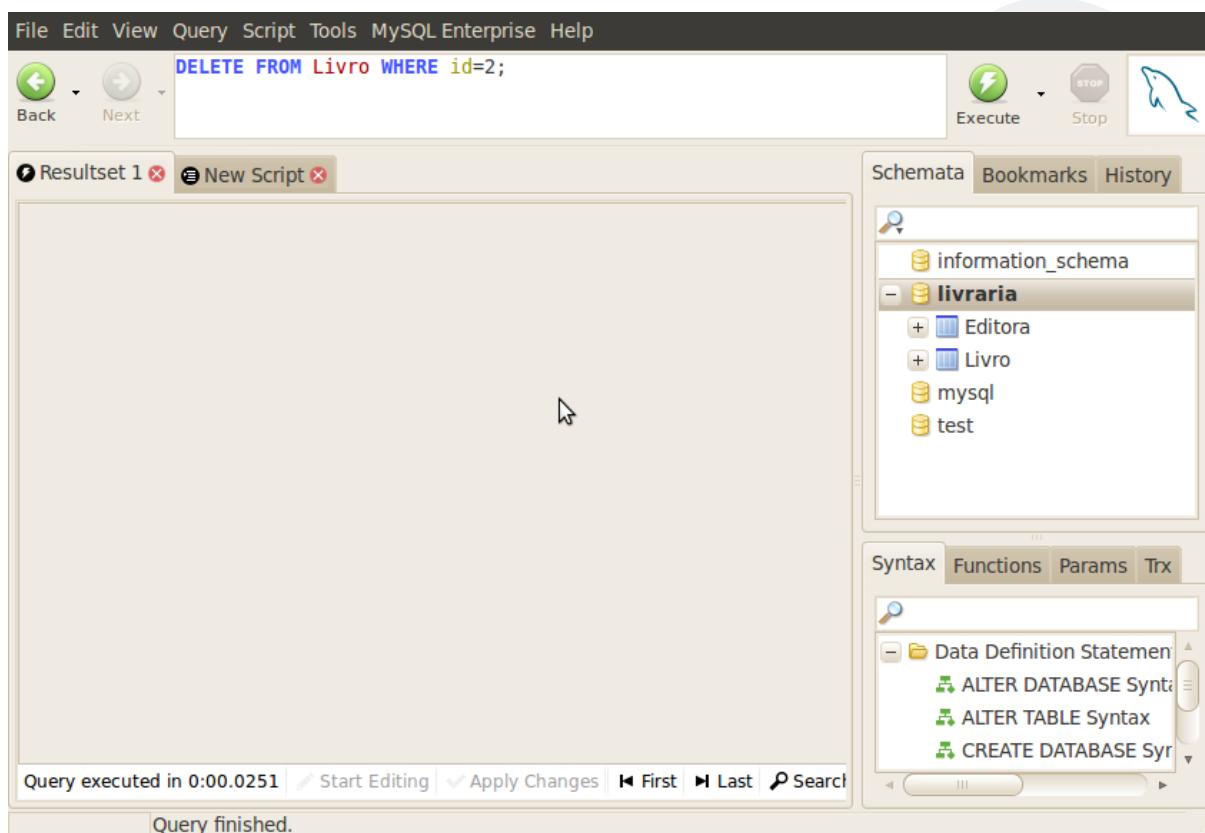
```
UPDATE Livro SET preco=92.9 WHERE id=1;
```

Below the query editor are buttons for Back, Next, Execute, and Stop. A results pane labeled "Resultset 1" is currently empty. To the right of the results pane is a sidebar titled "Schemata" which lists the databases: information_schema, livraria, mysql, and test. Under the "livraria" database, the tables Editora and Livro are visible. At the bottom of the interface, a message states "Query finished." and provides execution statistics: "Query executed in 0:00.0210".

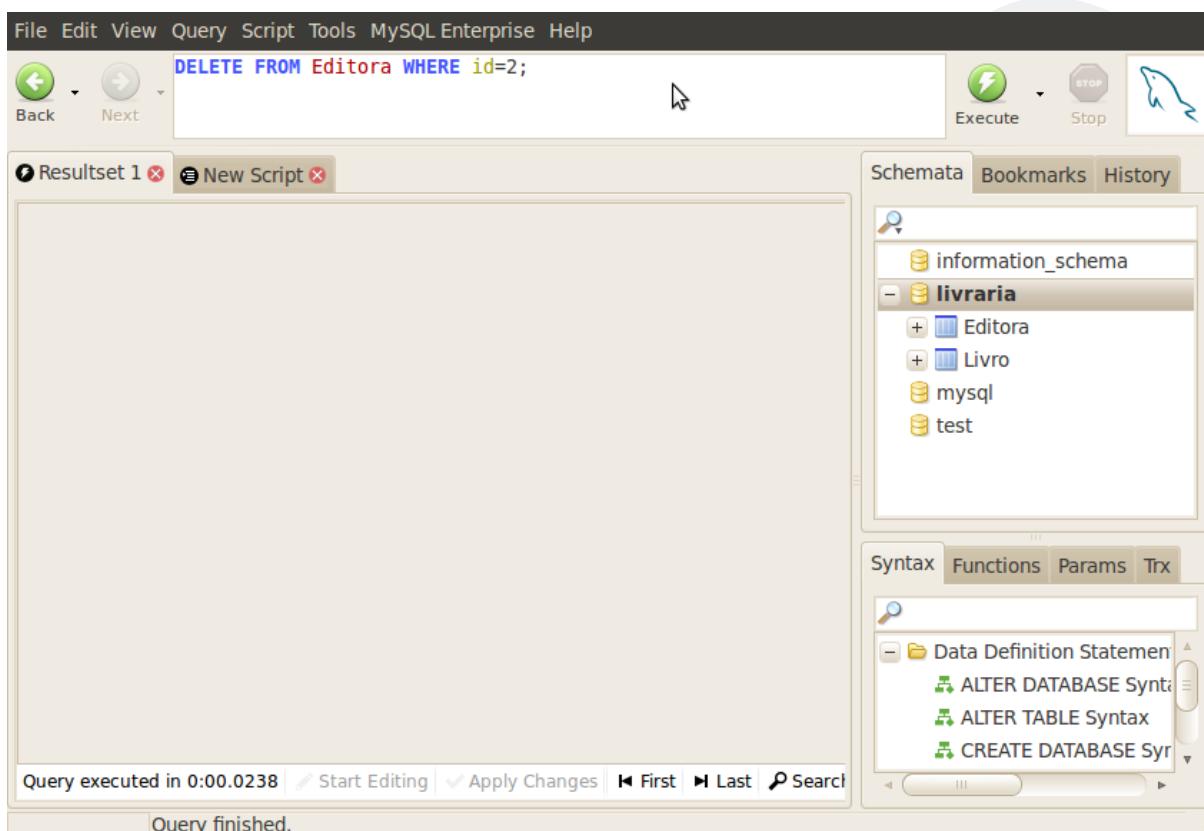
- 11 Altere alguns dos registros da tabela Editora. Veja o exemplo abaixo.



- 12 Remova alguns registros da tabela Livro. Veja o exemplo abaixo.



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



- 14 Faça uma consulta para buscar todos os livros associados as suas respectivas editoras. Veja um exemplo na figura abaixo.

The screenshot shows the MySQL Workbench interface. At the top, there's a menu bar with File, Edit, View, Query, Script, Tools, MySQL Enterprise, and Help. Below the menu is a toolbar with Back, Next, Execute, and Stop buttons. The main area contains a query editor window with the following SQL code:

```
SELECT * FROM Livro as L, Editora as E
WHERE L.editora_id=E.id;
```

Below the query editor is a Resultset window titled "Resultset 1" showing the results of the query:

	id	titulo	preco	editora_id	id	nome	email
1	Aprendendo C#	92.9	1	1	O'Reilly	oreilly@email.com	
3	JSF 2 Avançado	149.9	3	3	Apress	apress@email.com	

At the bottom of the Resultset window, it says "2 rows fetched in 0:00.0165". To the right of the Resultset window is a "Schemata" panel showing the database structure:

- information_schema
- livraria
 - + Editora
 - + Livro
- mysql
- test

Below the Schemata panel is a "Syntax" panel with tabs for Functions, Params, and Trx. The Functions tab is selected, showing options like Data Definition Statements, ALTER DATABASE Syntax, ALTER TABLE Syntax, and CREATE DATABASE Syntax.



JDBC

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 MySQL Server 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.

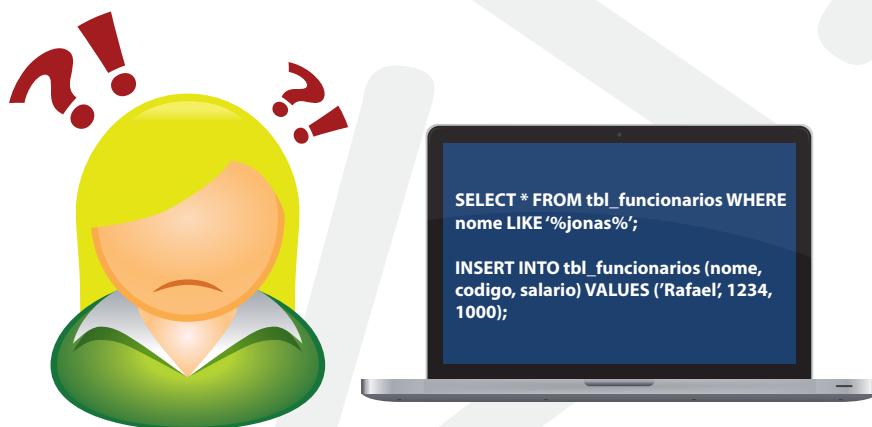


Figura 2.1: Usuários comuns não possuem conhecimento sobre SQL ou sobre o modelo relacional

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.

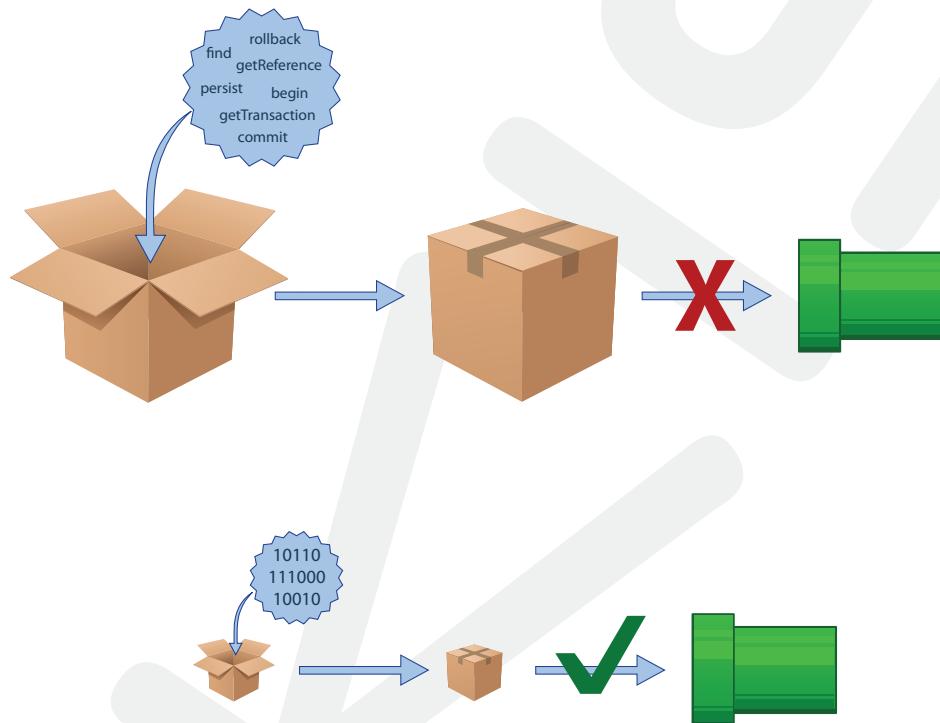


Figura 2.3: Diminuindo o tamanho das mensagens para não sobrecarregar o meio de comunicação

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.

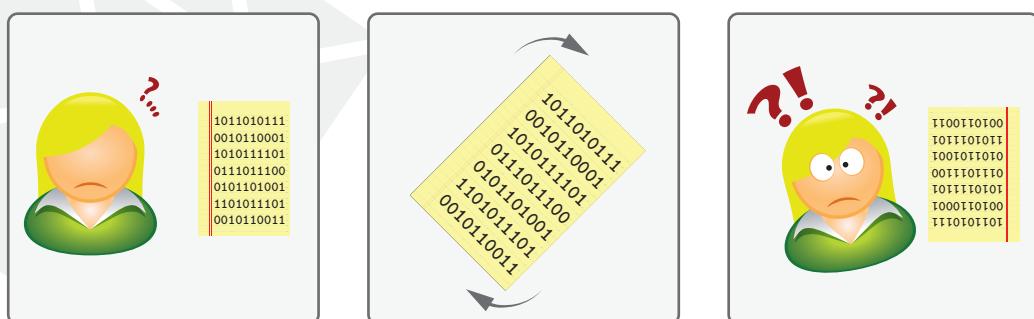


Figura 2.4: Mensagens binárias são altamente complexas para os seres humanos

Para resolver esse problema e facilitar o desenvolvimento das aplicações, as empresas proprietárias 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.



JDBC

Suponha que os drivers de conexão fossem desenvolvidos sem nenhum padrão. Cada driver teria sua própria interface, ou seja, seu próprio conjunto de instruções. Consequentemente, os desenvolvedores teriam de conhecer a interface de cada um dos drivers dos respectivos SGDBs que fossem utilizar.

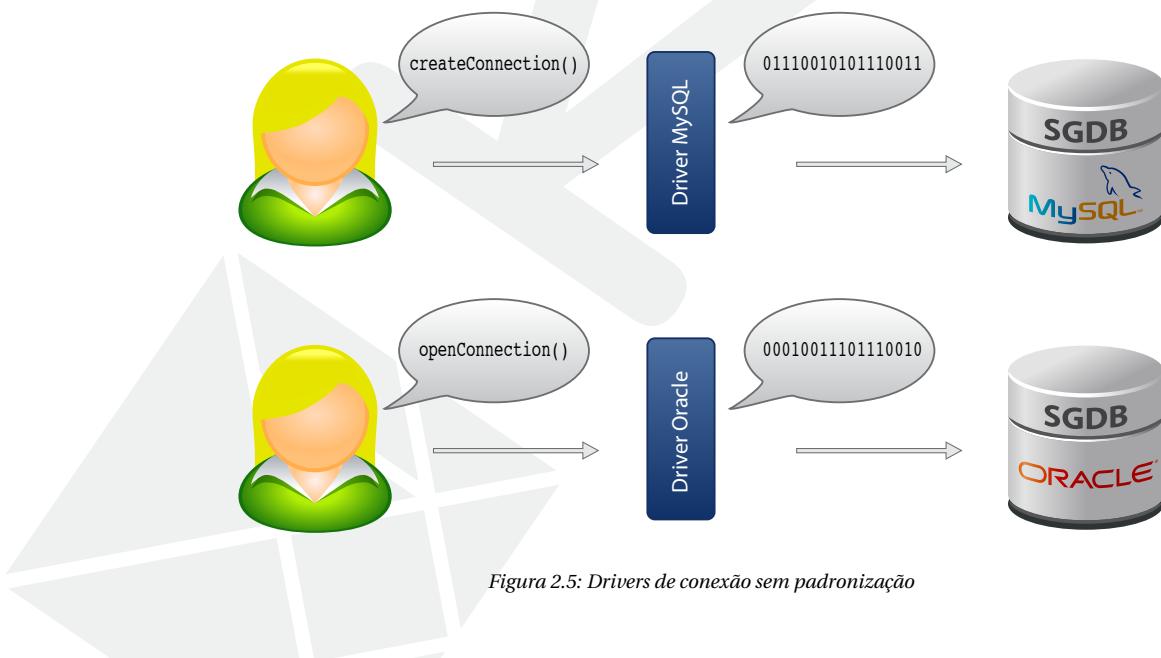


Figura 2.5: Drivers de conexão sem padronização

Para facilitar o trabalho do desenvolvedor da aplicação, a plataforma Java possui uma especificação que padroniza os drivers de conexão. A sigla dessa especificação é **JDBC** (*Java Database Connectivity*). Em geral, as empresas proprietárias dos SGDBs desenvolvem e distribuem drivers de conexão que seguem a especificação JDBC.

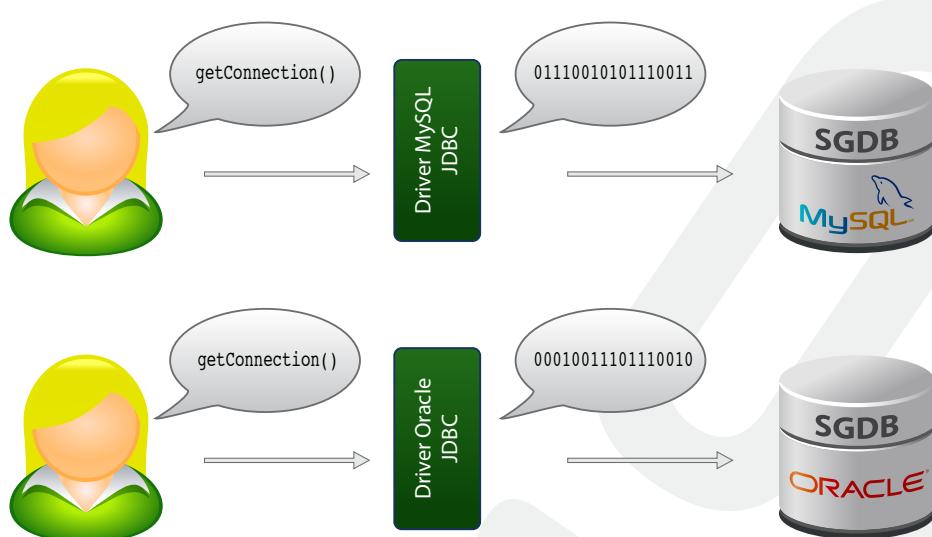


Figura 2.6: Drivers de conexão padronizados pela especificação JDBC



Instalando o Driver JDBC do MySQL Server

Podemos obter um driver de conexão JDBC para o MySQL Server na seguinte url:

<http://www.mysql.com/downloads/connector/j/>.

A instalação desse driver consiste em descompactar o arquivo obtido no site acima e depois incluir o arquivo **jar** com o driver no **class path** da aplicação.



Criando uma conexão

Com o driver de conexão JDBC adicionado à aplicação, já é possível criar uma conexão. Abaixo, estão as informações necessárias para a criação de uma conexão JDBC.

- Nome do driver JDBC.
- Endereço (IP e porta) do SGDB.
- Nome da base de dados.
- Um usuário do SGBD.
- Senha do usuário.

O nome do driver JDBC, o endereço do SGDB e nome da base de dados são definidos na **string de conexão** ou **url de conexão**. Veja o exemplo abaixo:

```
1 String stringDeConexao = "jdbc:mysql://localhost/livraria";
```

Código Java 2.1: String de conexão

A classe responsável pela criação de uma conexão JDBC é a `DriverManager` do pacote `java.sql`. A string de conexão, o usuário e a senha devem ser passados ao método estático `getConnection()` da classe `DriverManager` para que ela possa criar uma conexão JDBC.

```

1 String urlDeConexao = "jdbc:mysql://localhost/livraria";
2 String usuario = "root";
3 String senha = "root";
4 try {
5     Connection conexao = DriverManager.getConnection(urlDeConexao, usuario, senha);
6 } catch (SQLException e) {
7     e.printStackTrace();
8 }
```

Código Java 2.2: Criando uma conexão JDBC



Inserindo registros

Após estabelecer uma conexão JDBC, podemos executar operações. A primeira operação que realizaremos é a inserção de registros em uma tabela. O primeiro passo para executar essa operação é definir o código SQL correspondente.

```

1 String sql = "INSERT INTO Editora (nome, email) VALUES ('K19',' contato@k19.com.br');"
```

Código Java 2.3: Código SQL correspondente à operação de inserção

O código SQL correspondente à operação que desejamos executar deve ser passado como parâmetro para o método `prepareStatement()` de uma conexão JDBC. Esse método criará um objeto que representa a operação que será executada. A operação poderá ser executada posteriormente através do método `execute()`.

```

1 // criando um prepared statement
2 PreparedStatement comando = conexao.prepareStatement(sql);
3
4 // executando o prepared statement
5 comando.execute();
6 comando.close();
```

Código Java 2.4: Criando um prepared statement



Importante

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

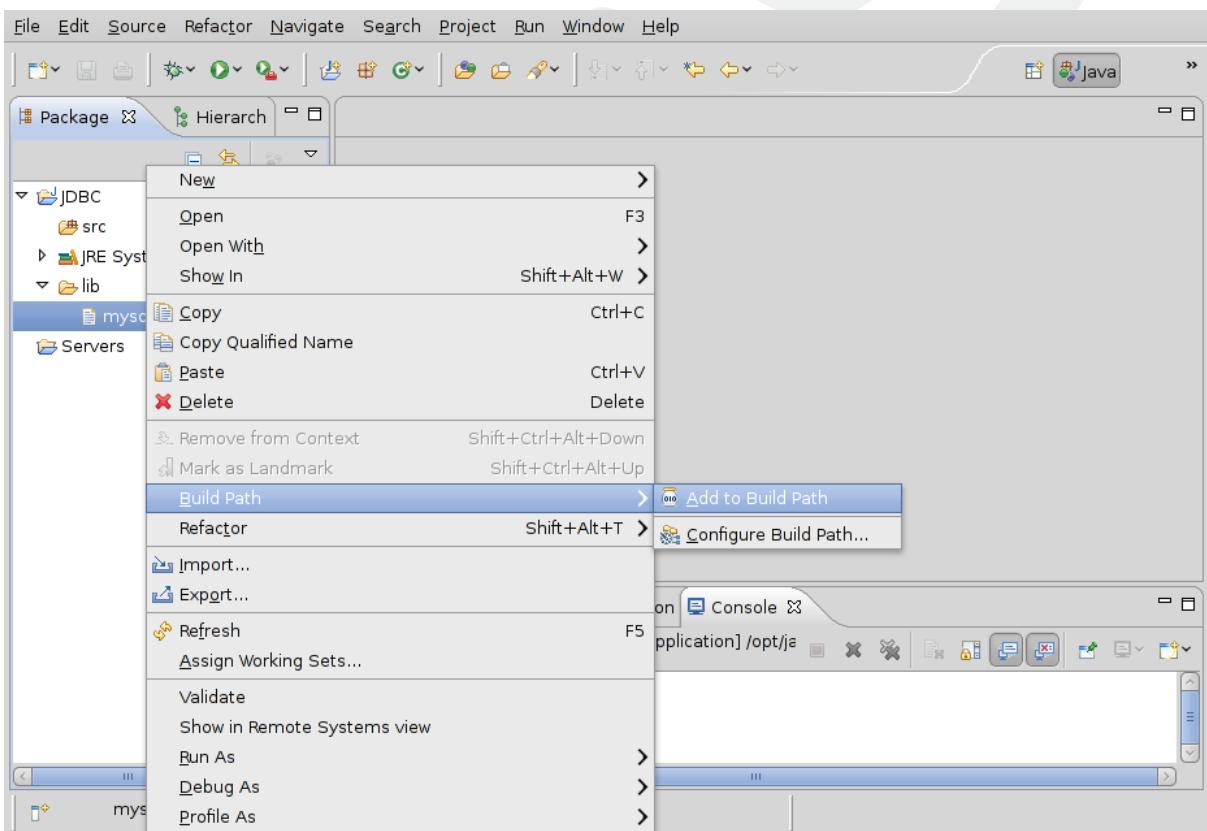
```
1 conexao.close();
```

Código Java 2.5: Finalizando uma conexão JDBC



Exercícios de Fixação

- 1 No Eclipse, crie um projeto Java chamado **JDBC**.
- 2 Crie uma pasta chamada **lib** no projeto **JDBC**.
- 3 Entre na pasta **K19-Arquivos/MySQL-Connector-JDBC** da Área de Trabalho e copie o arquivo **mysql-connector-java-5.1.13-bin.jar** para a pasta **lib** do projeto **JDBC**.
- 4 Adicione o arquivo **mysql-connector-java-5.1.13-bin.jar** ao **build path**. Veja a imagem abaixo.



- 5 Crie uma classe chamada **InsereEditora**, e adicione o seguinte conteúdo ao arquivo:

```
1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.PreparedStatement;
4 import java.util.Scanner;
5
6 public class InsereEditora {
7     public static void main(String[] args) {
8         String stringDeConexao = "jdbc:mysql://localhost:3306/livraria";
9         String usuario = "root";
10        String senha = "root";
11
12        Scanner entrada = new Scanner(System.in);
```

```

13
14     try {
15         System.out.println("Abrindo conexão...");
16         Connection conexao =
17             DriverManager.getConnection(stringDeConexao, usuario, senha);
18
19         System.out.println("Digite o nome da editora: ");
20         String nome = entrada.nextLine();
21
22         System.out.println("Digite o email da editora: ");
23         String email = entrada.nextLine();
24
25         String sql = "INSERT INTO Editora (nome, email) " +
26             "VALUES ('" + nome + "', '" + email + "')";
27
28         PreparedStatement comando = conexao.prepareStatement(sql);
29
30         System.out.println("Executando comando...");
31         comando.execute();
32
33         System.out.println("Fechando conexão...");
34         conexao.close();
35     } catch (Exception e) {
36         e.printStackTrace();
37     }
38 }
39 }
```

Código Java 2.6: InsereEditora.java

Execute e verifique se o registro foi inserido com sucesso na base de dados.



Exercícios Complementares

- 1 Crie uma classe chamada **InsereLivros** para cadastrar livros na base de dados.



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 Editora (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 MySQL Server. Para resolver esse problema manualmente, devemos adicionar o caractere “\” antes do caractere aspas simples que faz parte do nome da editora. Na sintaxe do

MySQL Server, o caractere “\” deve ser acrescentado imediatamente antes de todo caractere especial que deve ser tratado como um caractere comum.

```
1 INSERT INTO Editora (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 JDBC. 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 // lendo as entradas do usuário
2 System.out.println("Digite o nome da editora: ");
3 String nome = entrada.nextLine();
4
5 System.out.println("Digite o email da editora: ");
6 String email = entrada.nextLine();
7
8 // código sql com marcadores para as entradas do usuário
9 String sql = "INSERT INTO Editora (nome, email) VALUES (?, ?)";
10
11 // criando um comando a partir do código SQL
12 PreparedStatement comando = conexao.prepareStatement(sql);
13
14 // adicionando as entradas do usuários no comando
15 // o processo de sanitização ocorre aqui
16 comando.setString(1, nome);
17 comando.setString(2, email);
```

Código Java 2.10: “Sanitizando” as entradas dos usuários

Observe que o código SQL foi definido com parâmetros através do caractere “?”. Antes de executar o comando, é necessário determinar os valores dos parâmetros. Essa tarefa pode ser realizada através do método `setString()`, que recebe o índice (posição) do parâmetro no código SQL e o valor correspondente. Esse método faz o tratamento dos caracteres especiais contidos nos valores de entrada do usuário de acordo com as regras do SGDB utilizado.



Exercícios de Fixação

6 Provoque um erro de SQL Injection na classe `InsereEditoras`. (Dica: tente entradas com aspas simples.)

7 Altere o código para eliminar o problema do SQL Injection.

```

1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.PreparedStatement;
4 import java.util.Scanner;
5
6 public class InsereEditora {
7     public static void main(String[] args) {
8
9         String stringDeConexao = "jdbc:mysql://localhost:3306/livraria";
10        String usuario = "root";
11        String senha = "root";
12
13        Scanner entrada = new Scanner(System.in);
14
15        try {
16            System.out.println("Abrindo conexão...");
17            Connection conexao =
18                DriverManager.getConnection(stringDeConexao, usuario, senha);
19
20            System.out.println("Digite o nome da editora: ");
21            String nome = entrada.nextLine();
22
23            System.out.println("Digite o email da editora: ");
24            String email = entrada.nextLine();
25
26            String sql = "INSERT INTO Editora (nome, email) VALUES (?, ?)";
27
28            PreparedStatement comando = conexao.prepareStatement(sql);
29            comando.setString(1, nome);
30            comando.setString(2, email);
31
32            System.out.println("Executando comando...");
33            comando.execute();
34
35            System.out.println("Fechando conexão...");
36            conexao.close();
37        } catch (Exception e) {
38            e.printStackTrace();
39        }
40    }
41 }
```

Código Java 2.11: InsereEditora.java



Exercícios Complementares

- 2 Provoque um erro de SQL Injection na classe InsereLivros. (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

O processo para executar um comando de consulta é bem parecido com o processo de inserir

registros. O primeiro passo é definir a consulta em SQL.

```

1 String sql = "SELECT * FROM Editora;";
2 PreparedStatement comando = conexao.prepareStatement(sql);
3 System.out.println("Executando comando...");
4 ResultSet resultado = comando.executeQuery();
5
6

```

Código Java 2.13: Realizando uma consulta.

A diferença é que para executar um comando de consulta é necessário utilizar o método executeQuery() ao invés do execute(). Esse método devolve um objeto da interface java.sql.ResultSet, que é responsável por armazenar os resultados da consulta.

Os dados contidos no ResultSet podem ser acessados através de métodos, como o getString, getInt, getDouble, etc, de acordo com o tipo do campo. Esses métodos recebem como parâmetro uma string referente ao nome da coluna correspondente.

```

1 int id = resultado.getInt("id"),
2 String nome = resultado.getString("nome"),
3 String email = resultado.getString("email");
4
5

```

Código Java 2.14: Recuperando os dados de um ResultSet

O código acima mostra como os campos do primeiro registro da consulta são recuperados. Para recuperar os dados dos outros registros é necessário avançar o ResultSet através do método next().

```

1 int id1 = resultado.getInt("id"),
2 String nome1 = resultado.getString("nome"),
3 String email1 = resultado.getString("email");
4
5 resultado.next();
6
7 int id2 = resultado.getInt("id"),
8 String nome2 = resultado.getString("nome"),
9 String email2 = resultado.getString("email");
10
11

```

Código Java 2.15: Avançando o ResultSet

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

```

1 while(resultado.next()) {
2     int id = resultado.getInt("id"),
3     String nome = resultado.getString("nome"),
4     String email = resultado.getString("email");
5 }
6
7

```

Código Java 2.16: Iterando os registros do ResultSet



Exercícios de Fixação

- 8 Insira algumas editoras utilizando a classe InsereEditora que você criou anteriormente.

- 9 Adicione uma nova classe ao projeto chamada `ListaEditoras`. O objetivo é listar as editoras que foram salvas no banco.

```

1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
5
6 public class ListaEditoras {
7     public static void main(String[] args) {
8         String stringDeConexao = "jdbc:mysql://localhost:3306/livraria";
9         String usuario = "root";
10        String senha = "root";
11
12        try {
13            System.out.println("Abrindo conexão...");
14            Connection conexao =
15                DriverManager.getConnection(stringDeConexao, usuario, senha);
16
17            String sql = "SELECT * FROM Editora;";
18
19            PreparedStatement comando = conexao.prepareStatement(sql);
20
21            System.out.println("Executando comando...");
22            ResultSet resultado = comando.executeQuery();
23
24            System.out.println("Resultados encontrados: \n");
25            while (resultado.next()) {
26                System.out.printf("%d : %s - %s\n",
27                    resultado.getInt("id"),
28                    resultado.getString("nome"),
29                    resultado.getString("email"));
30            }
31
32            System.out.println("\nFechando conexão...");
33            conexao.close();
34        } catch (Exception e) {
35            e.printStackTrace();
36        }
37    }
38 }
```

Código Java 2.17: `ListaEditoras.java`



Exercícios Complementares

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

Connection Factory

Você deve ter percebido que em diversos pontos diferentes da nossa aplicação precisamos de uma conexão JDBC. Se a url de conexão for definida em cada um desses pontos, teremos um problema de manutenção. Imagine que o driver do banco seja atualizado ou que o IP do SGBD seja alterado. Teríamos que alterar o código da nossa aplicação em muitos lugares. Mais precisamente, cada ocorrência da url de conexão precisaria ser modificada. A probabilidade de algum ponto não ser corrigido é grande.

Para diminuir o trabalho de manutenção, podemos implementar uma classe responsável pela

criação e distribuição de conexões. A url de conexão deve ser definida nessa classe e somente nela. Consequentemente, alterações nas informações contidas na url de conexão afetariam apenas uma classe da aplicação.

```
1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.SQLException;
4
5 public class ConnectionFactory {
6     public static Connection createConnection() {
7         String stringDeConexao = "jdbc:mysql://localhost:3306/livraria";
8         String usuario = "root";
9         String senha = "root";
10
11     Connection conexao = null;
12
13     try {
14         conexao = DriverManager.getConnection(stringDeConexao, usuario, senha);
15     } catch (SQLException e) {
16         e.printStackTrace();
17     }
18     return conexao;
19 }
20 }
```

Código Java 2.19: ConnectionFactory.java

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



Exercícios de Fixação

- 10 Adicione uma nova classe chamada `ConnectionFactory` com o código abaixo.

```
1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.SQLException;
4
5 public class ConnectionFactory {
6     public static Connection createConnection() {
7         String stringDeConexao = "jdbc:mysql://localhost:3306/livraria";
8         String usuario = "root";
9         String senha = "root";
10
11     Connection conexao = null;
12
13     try {
14         conexao = DriverManager.getConnection(stringDeConexao, usuario, senha);
15     } catch (SQLException e) {
16         e.printStackTrace();
17     }
18     return conexao;
19 }
20 }
```

Código Java 2.20: ConnectionFactory.java

- 11** Altere as classes `InsereEditora` e `ListaEditoras` para que elas utilizem a fábrica de conexão. Depois, execute-as novamente.

```

1 import java.sql.Connection;
2 import java.sql.PreparedStatement;
3 import java.util.Scanner;
4
5 public class InsereEditora {
6     public static void main(String[] args) {
7         Scanner entrada = new Scanner(System.in);
8
9         try {
10             System.out.println("Abrindo conexão...");
11             Connection conexao = ConnectionFactory.createConnection();
12
13             System.out.println("Digite o nome da editora: ");
14             String nome = entrada.nextLine();
15
16             System.out.println("Digite o email da editora: ");
17             String email = entrada.nextLine();
18
19             String sql = "INSERT INTO Editora (nome, email) " +
20                     "VALUES (?, ?)";
21
22             PreparedStatement comando = conexao.prepareStatement(sql);
23             comando.setString(1, nome);
24             comando.setString(2, email);
25
26             System.out.println("Executando comando...");
27             comando.execute();
28
29             System.out.println("Fechando conexão...");
30             conexao.close();
31         } catch (Exception e) {
32             e.printStackTrace();
33         }
34     }
35 }
```

Código Java 2.21: `InsereEditora.java`

```

1 import java.sql.Connection;
2 import java.sql.PreparedStatement;
3 import java.sql.ResultSet;
4
5 public class ListaEditoras {
6     public static void main(String[] args) {
7         try {
8             System.out.println("Abrindo conexão...");
9             Connection conexao = ConnectionFactory.createConnection();
10
11             String sql = "SELECT * FROM Editora;";
12
13             PreparedStatement comando = conexao.prepareStatement(sql);
14
15             System.out.println("Executando comando...");
16             ResultSet resultado = comando.executeQuery();
17
18             System.out.println("Resultados encontrados: \n");
19             while (resultado.next()) {
20                 System.out.printf("%d : %s - %s\n",
21                     resultado.getInt("id"),
22                     resultado.getString("nome"),
23                     resultado.getString("email"));
24             }
25
26             System.out.println("\nFechando conexão...");
27             conexao.close();
28 }
```

```
28 } catch (Exception e) {  
29     e.printStackTrace();  
30 }  
31 }  
32 }
```

Código Java 2.22: *ListaEditoras.java*



Exercícios Complementares

- 6 Altere as classes `InsereLivro` e `ListaLivros` para que elas utilizem a fábrica de conexão. Depois, execute-as novamente.



Desafios

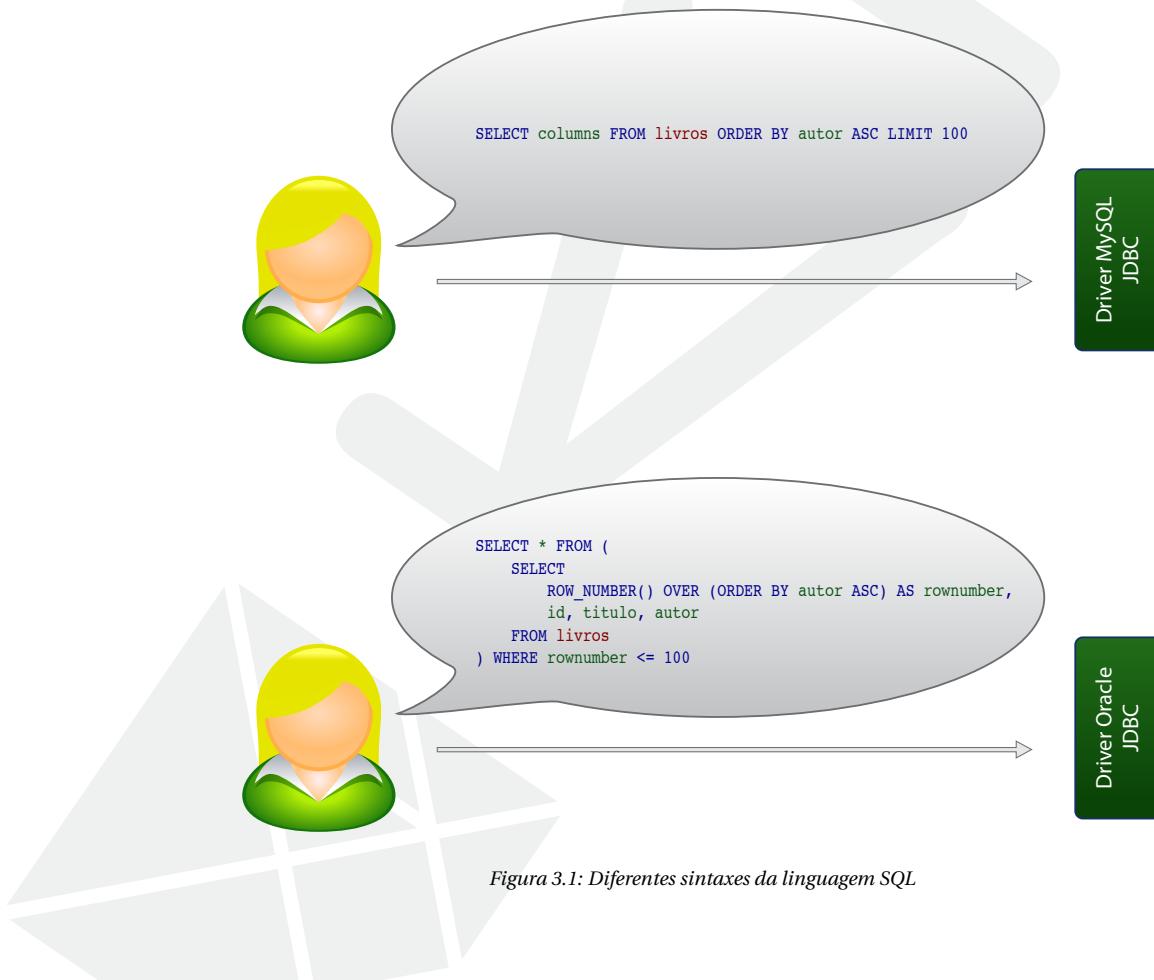
- 1 Implemente a remoção de editoras pelo id.
- 2 Implemente a alteração dos dados das editoras pelo id.

JPA 2 E HIBERNATE



Múltiplas sintaxes da linguagem SQL

No capítulo anterior, utilizamos a especificação JDBC para fazer uma aplicação Java 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.



Orientação a Objetos VS Modelo Relacional

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

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.

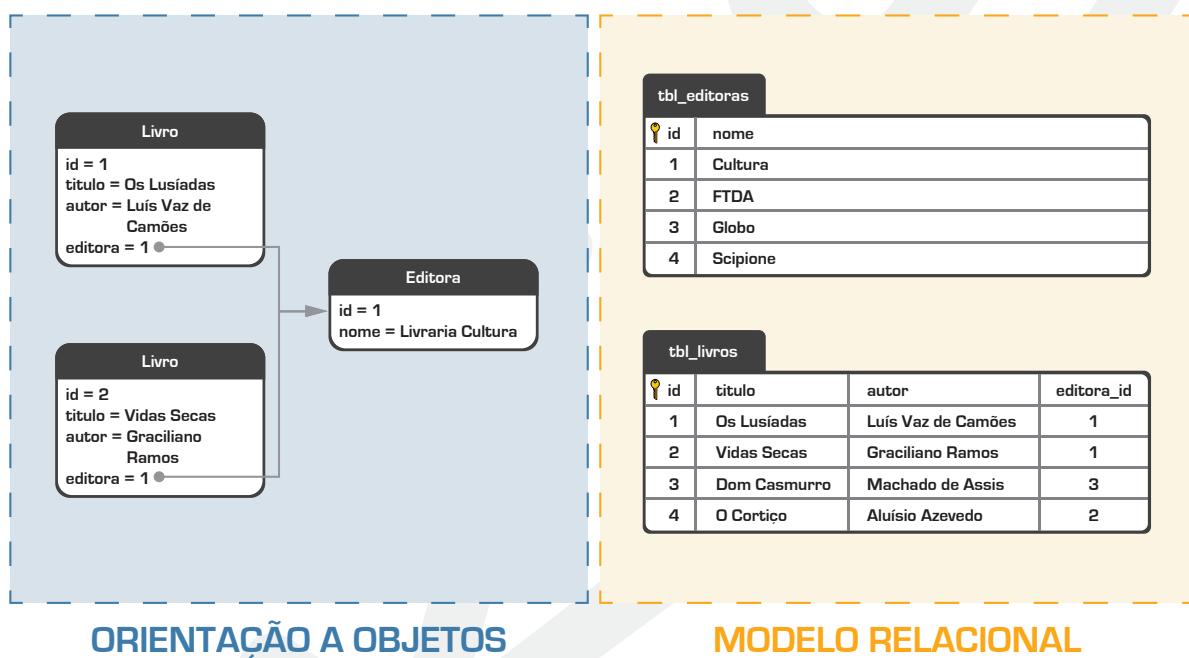


Figura 3.2: Modelo Orientado a Objetos vs Modelo Relacional



Ferramentas ORM

Para facilitar a comunicação entre aplicações Java 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. A principal ferramenta ORM para Java utilizada no mercado é o Hibernate. Mas, existem outras que possuem o mesmo objetivo.



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



O que é JPA e Hibernate?

Após o sucesso do Hibernate, a especificação **JPA** (*Java Persistence API*) foi criada com o objetivo de padronizar as ferramentas ORM para aplicações Java e consequentemente diminuir a complexidade do desenvolvimento. Atualmente, essa especificação está na sua segunda versão.

Ela especifica um conjunto de classes e métodos que as ferramentas ORM devem implementar. Veja que a JPA é apenas uma especificação. Ela não implementa nenhum código. Para isso, utilizamos alguma das diversas implementações da JPA. Neste curso, utilizaremos o Hibernate como implementação de JPA. As outras implementações de JPA mais conhecidas são **EclipseLink** e **OpenJPA**. Optamos por utilizar o Hibernate por ele ser o mais antigo e mais utilizado atualmente.

Caso você queira utilizar outra ferramenta ORM, poderá aplicar os conceitos aqui aprendidos justamente porque eles seguem a mesma especificação. Assim, podemos programar voltado à especificação e substituir uma implementação pela outra, sem precisar reescrever o código da nossa aplicação. Claro que teríamos que alterar alguns arquivos de configuração, mas o código da aplicação permaneceria o mesmo.



Bibliotecas

Antes de começar a utilizar o Hibernate, é necessário baixar do site oficial o **bundle** que inclui os jar's do hibernate e todas as suas dependências. Neste curso, utilizaremos a versão 3.5.1. A url do site oficial do Hibernate é <http://www.hibernate.org/>.



Configuração

Para configurar o Hibernate em uma aplicação, devemos criar um arquivo chamado **persistence.xml**. O conteúdo desse arquivo possuirá informações sobre o banco de dados, como a url de conexão, usuário e senha, além de dados sobre a implementação de JPA que será utilizada.

O arquivo **persistence.xml** deve estar em uma pasta chamada **META-INF**, que deve estar no classpath da aplicação. Veja abaixo um exemplo de configuração para o **persistence.xml**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0"
3   xmlns="http://java.sun.com/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/←
6     ns/persistence/persistence_2_0.xsd">
7 
8   <persistence-unit name="K19-PU" transaction-type="RESOURCE_LOCAL">
9     <provider>org.hibernate.ejb.HibernatePersistence</provider>
10    <properties>
11      <property name="hibernate.dialect" value="org.hibernate.dialect.←
12        MySQL5InnoDBialect"/>
13 
14      <property name="hibernate.hbm2ddl.auto" value="create"/>
15 
16      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
17 
18      <property name="javax.persistence.jdbc.user" value="usuario"/>
19 
20      <property name="javax.persistence.jdbc.password" value="senha"/>
21 
22      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/←
23        K19-DB"/>
24    </properties>
25  </persistence-unit>
26 </persistence>
```

Código XML 3.1: persistence.xml

A propriedade **hibernate.dialect** permite que a aplicação escolha qual sintaxe de SQL deve ser utilizada pelo Hibernate.



Mais Sobre

Consulte o artigo da K19 sobre configuração do Hibernate e MySQL na seguinte url:
<http://www.k19.com.br/artigos/configurando-hibernate-com-mysql/>.



Mapeamento

Um dos principais objetivos das ferramentas ORM é estabelecer o mapeamento entre os conceitos do modelo orientado a objetos e os conceitos do modelo relacional. Esse mapeamento pode ser definido através de XML ou de maneira mais prática com anotações Java.

A seguir, veremos as principais anotações Java de mapeamento do JPA. Essas anotações estão no pacote **javax.persistence**.

@Entity É a principal anotação do JPA. Ela deve aparecer antes do nome de uma classe e deve ser definida em todas as classes que terão objetos persistidos no banco de dados.

As classes anotadas com @Entity são mapeadas para tabelas. Por convenção, as tabelas possuem os mesmos nomes das classes. Mas, podemos alterar esse comportamento utilizando a anotação @Table.

Os atributos declarados em uma classe anotada com @Entity são mapeados para colunas na tabela correspondente à classe. Outra vez, por convenção, as colunas possuem os mesmos nomes dos atributos. Novamente, podemos alterar esse padrão utilizando a anotação @Column.

@Id Utilizada para indicar qual atributo de uma classe anotada com @Entity será mapeado para a chave primária da tabela correspondente à classe. Geralmente o atributo anotado com @Id é do tipo Long.

@GeneratedValue Geralmente vem acompanhado da anotação @Id. Serve para indicar que o atributo é gerado pelo banco, no momento em que um novo registro é inserido.

@Table Utilizada para alterar o nome padrão da tabela. Ela recebe o parâmetro name para indicar qual nome deve ser utilizado na tabela. Veja o exemplo:

```

1 @Table(name="Publisher")
2 @Entity
3 public class Editora {
4     // ...
5 }
```

Código Java 3.1: Editora.java

@Column Utilizado para alterar o nome da coluna que será usado na tabela. Caso você esteja utilizando um banco de dados legado, no qual os nomes das colunas já foram definidos, você pode mudar através dessa anotação. Além disso, podemos estabelecer certas restrições, como determinar se o campo pode ou não ser nulo.

```

1 @Entity
2 public class Editora {
3     @Column(name="publisher_name", nullable=false)
4     private String nome;
5 }
6 }
```

Código Java 3.2: Editora.java

@Transient Serve para indicar que um atributo não deve ser persistido, ou seja, os atributos anotados com @Transient não são mapeados para colunas.

@Lob Utilizado para atributos que armazenam textos muito grandes, ou arquivos binários contendo imagens ou sons que serão persistidos no banco de dados. O tipo do atributo deve ser String, Byte[], byte[] ou java.sql.Blob.

@Temporal Utilizado para atributos do tipo Calendar ou Date. Por padrão, tanto data quanto hora são armazenados no banco de dados. Mas, com a anotação @Temporal, podemos mandar persistir somente a data ou somente a hora.

```

1 @Entity
2 public class Livro {
3     @Temporal(TemporalType.DATE)
4     private Calendar publicacao;
```

```
5 |     // ...
6 | }
7 | }
```

Código Java 3.3: Livro.java



Gerando Tabelas

Uma das vantagens de se utilizar o Hibernate é que ele é capaz de gerar as tabelas do banco para a nossa aplicação. Ele faz isso de acordo com as anotações colocadas nas classes e as informações presentes no `persistence.xml`.

As tabelas são geradas através de um método da classe Persistence, o `createEntityManagerFactory(String entityUnit)`. O parâmetro `entityUnit` permite escolher, pelo nome, uma unidade de persistência definida no `persistence.xml`.

A política de criação das tabelas pode ser alterada modificando o valor a propriedade `hibernate.hbm2ddl.auto` no arquivo `persistence.xml`. Podemos, por exemplo, fazer com que o Hibernate sempre sobrescreva as tabelas existentes, bastando definir a propriedade `hibernate.hbm2ddl.auto` com o valor `create-drop`.

```
1 <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
```

Uma outra opção é configurar o Hibernate para simplesmente atualizar as tabelas de acordo com as mudanças nas anotações sem removê-las. Nesse caso, o valor da propriedade `hibernate.hbm2ddl.auto` deve ser `update`.

```
1 <property name="hibernate.hbm2ddl.auto" value="update"/>
```



Exercícios de Fixação

- 1 Crie um projeto no Eclipse chamado JPA2-Hibernate e feche o projeto JDBC para não gerar confusão na hora de manipular os arquivos.
- 2 Crie uma pasta chamada `lib` dentro do projeto JPA2-Hibernate.
- 3 Entre na pasta K19-Arquivos/Hibernate da Área de Trabalho e copie os jar's do Hibernate para a pasta `lib` do projeto JPA2-Hibernate.
- 4 Entre na pasta K19-Arquivos/MySQL-Connector-JDBC da Área de Trabalho e copie o arquivo `mysql-connector-java-5.1.13-bin.jar` para pasta `lib` do projeto JPA2-Hibernate.
- 5 Entre na pasta K19-Arquivos/SLF4J da Área de Trabalho e copie os jar's para pasta `lib` do

projeto JPA2-Hibernate.

- 6 Entre na pasta K19-Arquivos/Log4J da Área de Trabalho e copie o arquivo log4j-1.2.16.jar para pasta lib do projeto JPA2-Hibernate.
- 7 Adicione os jar's da pasta lib ao build path do projeto JPA2-Hibernate. Você deve selecionar os arquivos e adicioná-los no build path.
- 8 Crie uma pasta chamada META-INF na pasta src no projeto JPA2-Hibernate.
- 9 Crie o arquivo de configurações persistence.xml na pasta META-INF.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0"
3   xmlns="http://java.sun.com/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/←
6     ns/persistence/persistence_2_0.xsd">
7
8   <persistence-unit name="livraria-pu" transaction-type="RESOURCE_LOCAL">
9     <provider>org.hibernate.ejb.HibernatePersistence</provider>
10    <properties>
11      <property name="hibernate.dialect" value="org.hibernate.dialect.←
12        MySQL5InnoDBDialect"/>
13
14      <property name="hibernate.hbm2ddl.auto" value="create"/>
15
16      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
17
18      <property name="javax.persistence.jdbc.user" value="root"/>
19
20      <property name="javax.persistence.jdbc.password" value="root"/>
21
22      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/←
23        livraria"/>
24    </properties>
25  </persistence-unit>
26 </persistence>

```

Código XML 3.4: persistence.xml

- 10 Crie uma classe para modelar as editoras da nossa livraria e acrescente as anotações necessárias para fazer o mapeamento. Obs: As anotações devem ser importadas do pacote javax.persistence.

```

1 @Entity
2 public class Editora {
3   @Id @GeneratedValue
4   private Long id;
5
6   private String nome;
7
8   private String email;
9
10  // GETTERS AND SETTERS
11 }

```

Código Java 3.4: Editora.java

- 11** Apague a tabela Livro e depois a Editora.
- 12** Configure o Log4J criando um arquivo chamado log4j.properties na pasta src do projeto JPA2-Hibernate.

```

1 log4j.rootCategory=INFO, CONSOLE
2 log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
3 log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
4 log4j.appender.CONSOLE.layout.ConversionPattern=%r [%t] %-5p %c - %m%n

```

Arquivo de Propriedades 3.1: log4j.properties

- 13** Gere as tabelas através da classe Persistence. Para isso, crie uma classe com método main. Obs: As classes devem ser importadas do pacote javax.persistence.

```

1 public class GeraTabelas {
2     public static void main(String[] args) {
3         EntityManagerFactory factory =
4             Persistence.createEntityManagerFactory("livraria-pu");
5
6         factory.close();
7     }
8 }

```

Código Java 3.5: GeraTabelas.java

Através do MySQL Query Browser, verifique se a tabela Editora foi criada corretamente.



Manipulando entidades

Para manipular as entidades da nossa aplicação, devemos utilizar um EntityManager que é obtido através de uma EntityManagerFactory.

```

1 EntityManagerFactory factory =
2     Persistence.createEntityManagerFactory("K19");
3
4 EntityManager manager = factory.createEntityManager();

```

Código Java 3.6: Obtendo um EntityManager

Persistindo

Para armazenar as informações de um objeto no banco de dados, o primeiro passo é utilizar o método persist() do EntityManager.

```

1 Editora novaEditora = new Editora();
2 novaEditora.setNome("K19 - Livros")
3 novaEditora.setEmail("contato@k19.com.br");
4
5 manager.persist(novaEditora);

```

Código Java 3.7: Marcando um objeto para ser persistido

É importante destacar que o método `persist()` apenas marca os objetos que devem ser armazenados no banco de dados. Os objetos serão armazenados após a chamada do método `commit()`, como veremos adiante.

Buscando

Para obter um objeto que contenha informações do banco de dados, podemos utilizar o método `find()` ou o método `getReference()` do `EntityManager`.

```
1 Editora editora1 = manager.find(Editora.class, 1L);
2 Editora editora2 = manager.getReference(Editora.class, 2L);
```

Código Java 3.8: Obtendo objetos com informações do banco de dados

Há uma diferença entre os dois métodos básicos de busca `find()` e `getReference()`. O método `find()` recupera os dados desejados imediatamente. Já o método `getReference()` posterga essa tarefa até a primeira chamada de um método `get` do objeto.

Removendo

Para remover o registro correspondente a um objeto, devemos utilizar o método `remove()` do `EntityManager`.

```
1 Editora editora1 = manager.find(Editora.class, 1L);
2 manager.remove(editora1);
```

Código Java 3.9: Marcando um objeto para ser removido

Atualizando

Para alterar os dados do registro correspondente a um objeto, basta utilizar os próprios métodos `setters` desse objeto.

```
1 Editora editora = manager.find(Editora.class, 1L);
2 editora.setNome("K19 - Livros e Publicações");
```

Código Java 3.10: Alterando os dados de um registro

Listando

Para obter uma listagem com todos os objetos referentes aos registros de uma tabela, podemos utilizar a linguagem de consulta do JPA, a **JPQL**, que é muito parecida com a linguagem SQL. A principal vantagem do JPQL em relação ao SQL é que a sintaxe do JPQL não depende do SGDB utilizado.

```
1 Query query = manager.createQuery("SELECT e FROM Editora e");
2 List<Editora> editoras = query.getResultList();
```

Código Java 3.11: Obtendo uma lista de objetos com informações do banco de dados

Transações

As modificações realizadas nos objetos administrados por um `EntityManager` são mantidas em memória. Em certos momentos, é necessário sincronizar os dados da memória com os dados do

banco de dados. Essa sincronização deve ser realizada através de uma transação JPA criada pelo EntityManager que administra os objetos que desejamos sincronizar.

Para abrir uma transação, utilizamos o método `begin()`.

```
1 manager.getTransaction().begin();
```

Código Java 3.12: Abrindo uma transação

Com uma transação aberta, podemos sincronizar os dados da memória com os dados do banco através do método `commit()`.

```
1 Editora editora = manager.find(Editora.class, 1L);
2 editora.setNome("K19 - Livros e Publicações");
3
4 manager.getTransaction().begin();
5 manager.getTransaction().commit();
```

Código Java 3.13: Sincronizando com o método commit()



Exercícios de Fixação

- 14 No arquivo de configurações `persistence.xml`, altere o valor da propriedade `hibernate.hbm2ddl.auto` para `update`. Assim, as tabelas não serão recriadas a cada execução e sim apenas atualizadas.
- 15 Crie um teste para inserir editoras no banco de dados.

```
1 public class InsereEditoraComJPA {
2
3     public static void main(String[] args) {
4         EntityManagerFactory factory =
5             Persistence.createEntityManagerFactory("livraria-pu");
6
7         EntityManager manager = factory.createEntityManager();
8
9         Editora novaEditora = new Editora();
10
11        Scanner entrada = new Scanner(System.in);
12
13        System.out.println("Digite o nome da editora: ");
14        novaEditora.setNome(entrada.nextLine());
15
16        System.out.println("Digite o email da editora: ");
17        novaEditora.setEmail(entrada.nextLine());
18
19        manager.persist(novaEditora);
20
21        manager.getTransaction().begin();
22        manager.getTransaction().commit();
23
24        manager.close();
25        factory.close();
26    }
27}
```

Código Java 3.14: InsereEditoraComJPA.java

- 16 Crie um teste para listar as editoras inseridas no banco de dados.

```

1 public class ListaEditorasComJPA {
2
3     public static void main(String[] args) {
4         EntityManagerFactory factory =
5             Persistence.createEntityManagerFactory("livraria-pu");
6
7         EntityManager manager = factory.createEntityManager();
8
9         Query query = manager.createQuery("SELECT e FROM Editora e");
10        List<Editora> editoras = query.getResultList();
11
12        for(Editora e : editoras) {
13            System.out.println("EDITORIA: " + e.getNome() + " - " + e.getEmail());
14        }
15
16        manager.close();
17        factory.close();
18    }
19 }
```

Código Java 3.15: ListaEditorasComJPA.java



Repository

A interface EntityManager do JPA oferece recursos suficientes para que os objetos do domínio sejam recuperados ou persistidos no banco de dados. Porém, em aplicações com alta complexidade e grande quantidade de código, “espalhar” as chamadas aos métodos do EntityManager 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 recuperar e para adicionar elementos.

Os repositórios podem trabalhar com objetos prontos na memória ou reconstrui-los com dados obtidos de um banco de dados. O acesso ao banco de dados pode ser realizado através de ferramentas ORM como o Hibernate.



Mais Sobre

O padrão Repository é semelhante ao padrão DAO – Data Access Object.

```

1 class EditoraRepository {
2     private EntityManager manager;
3
4     public EditoraRepository(EntityManager manager) {
5         this.manager = manager;
6     }
7
8     public void adiciona(Editora e) {
9         this.manager.persist(e);
10    }
11    public Editora busca(Long id) {
```

```

12     return this.manager.find(Editora.class, id);
13 }
14 public List<Editora> buscaTodas() {
15     Query query = this.manager.createQuery("SELECT e FROM Editora e");
16     return query.getResultList();
17 }
18 }
```

Código Java 3.16: *EditoraRepository.java*

```

1 EntityManagerFactory factory = Persistence.createEntityManagerFactory("K19");
2 EntityManager manager = factory.createEntityManager();
3 EditoraRepository editoraRepository = new EditoraRepository(manager);
4
5 List<Editora> editoras = editoraRepository.buscaTodas();
```

Código Java 3.17: Utilizando um repositório



Exercícios de Fixação

- 17** Implemente um repositório de editoras criando uma nova classe no projeto JPA2-Hibernate.

```

1 class EditoraRepository {
2     private EntityManager manager;
3
4     public EditoraRepository(EntityManager manager) {
5         this.manager = manager;
6     }
7
8     public void adiciona(Editora e) {
9         this.manager.persist(e);
10 }
11 public Editora busca(Long id) {
12     return this.manager.find(Editora.class, id);
13 }
14 public List<Editora> buscaTodas() {
15     Query query = this.manager.createQuery("SELECT e FROM Editora e");
16     return query.getResultList();
17 }
18 }
```

Código Java 3.18: *EditoraRepository.java*

- 18** Altere a classe *InsereEditoraComJPA* para que ela utilize o repositório de editoras.

```

1 public class InsereEditoraComJPA {
2
3     public static void main(String[] args) {
4         EntityManagerFactory factory =
5             Persistence.createEntityManagerFactory("livraria-pu");
6
7         EntityManager manager = factory.createEntityManager();
8
9         EditoraRepository editoraRepository = new EditoraRepository(manager);
10
11         Editora novaEditora = new Editora();
12 }
```

```

13     Scanner entrada = new Scanner(System.in);
14
15     System.out.println("Digite o nome da editora: ");
16     novaEditora.setNome(entrada.nextLine());
17
18     System.out.println("Digite o email da editora: ");
19     novaEditora.setEmail(entrada.nextLine());
20
21     editoraRepository.adiciona(novaEditora);
22
23     manager.getTransaction().begin();
24     manager.getTransaction().commit();
25
26     manager.close();
27     factory.close();
28 }
29 }
```

Código Java 3.19: InsereEditoraComJPA.java

- 19 Altere a classe ListaEditorasComJPA para que ela utilize o repositório de editoras.

```

1 public class ListaEditorasComJPA {
2
3     public static void main(String[] args) {
4         EntityManagerFactory factory =
5             Persistence.createEntityManagerFactory("livraria-pu");
6
7         EntityManager manager = factory.createEntityManager();
8
9         EditoraRepository editoraRepository = new EditoraRepository(manager);
10
11     List<Editora> editoras = editoraRepository.buscaTodas();
12
13     for(Editora e : editoras) {
14         System.out.println("EDITORA: " + e.getNome() + " - " + e.getEmail());
15     }
16
17     manager.close();
18     factory.close();
19 }
20 }
```

Código Java 3.20: ListaEditorasComJPA.java



WEB CONTAINER



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 aplicação 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.

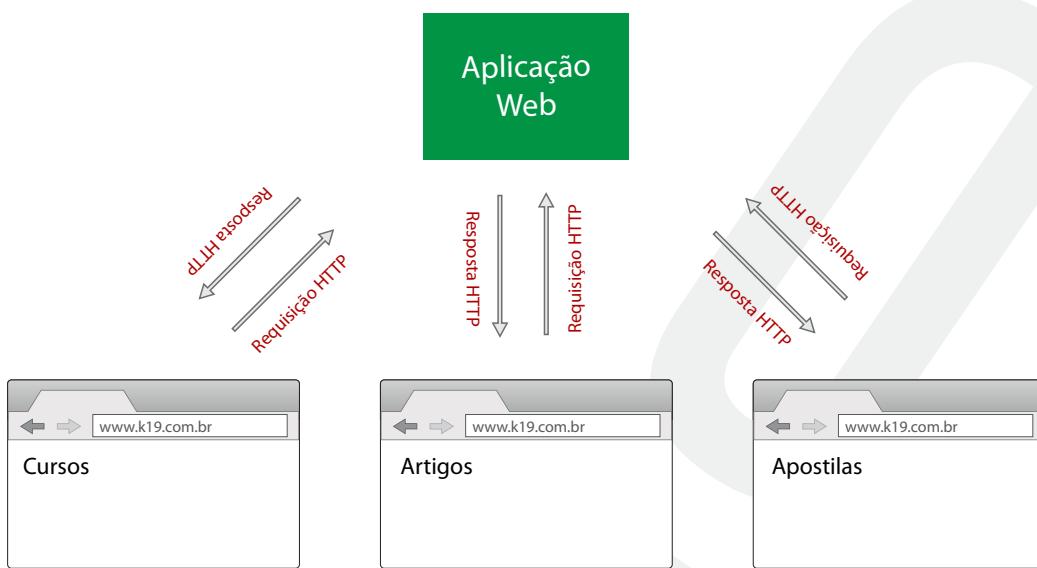


Figura 4.1: Necessidades de uma aplicação web

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 Java oferece uma solução genérica que pode ser utilizada para desenvolver aplicações web. Conheceremos essa solução a seguir.

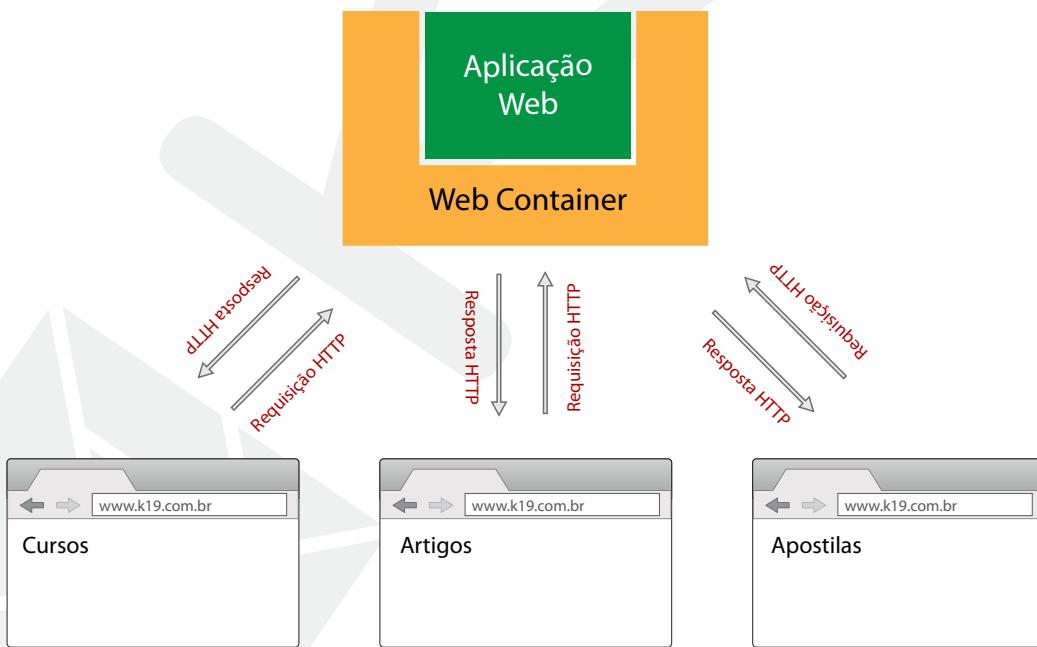


Figura 4.2: Web Container



Web Container

Uma aplicação web Java deve ser implantada em um **Web Container** para obter os recursos fundamentais que ela necessita. Um Web Container é responsável:

- Pelo envio e recebimento de mensagens HTTP.
- Por permitir que as aplicações sejam acessadas simultaneamente por vários usuários de uma maneira eficiente.
- Por permitir que as páginas de uma aplicação web sejam geradas dinamicamente.

Os dois Web Containers mais importantes do mercado são **Tomcat** e **Jetty**. Também podemos utilizar um servidor de aplicação Java EE como o **JBoss**, **Glassfish** ou **WebSphere**, pois eles possuem um Web Container internamente.



Servlet e Java EE

Como é comum na plataforma Java, foi definida uma especificação para padronizar a interface dos recursos oferecidos pelos Web Containers. Essa especificação é chamada **Servlet** e atualmente está na versão 3. Para consultá-la, acesse <http://jcp.org/en/jsr/detail?id=315>.

A especificação Servlet faz parte do **Java EE**. O Java EE é uma especificação que agrupa diversas outras especificações. Para consultá-la, acesse <http://jcp.org/en/jsr/detail?id=316>.

Apesar das especificações, os Web Containers possuem algumas diferenças nas configurações que devem ser realizadas pelos desenvolvedores. Dessa forma, não há 100% de portabilidade. Contudo, a maior parte das configurações e do modelo de programação é padronizado. Sendo assim, se você conhece bem um dos Web Containers, também conhece bastante dos outros.

Neste treinamento, optamos pela utilização do servidor de aplicação Glassfish 3.0. Esse servidor segue a especificação Java EE 6 e portanto contém um Web Container. Mostraremos, a seguir, a sua instalação e configuração.



Mais Sobre

Consulte os artigos da K19 sobre instalação e configuração do Glassfish para mais detalhes.

<http://www.k19.com.br/artigos/installando-glassfish/>

<http://www.k19.com.br/artigos/configurando-o-glassfish-no-ide-eclipse/>



Exercícios de Fixação

- 1 Na Área de Trabalho, entre na pasta K19-Arquivos e copie o arquivo `glassfish-3.0.1-with-hibernate.zip` para o seu Desktop. Descompacte-o na própria Área de Trabalho.

- 2 Ainda na Área de Trabalho, entre na pasta glassfishv3/glassfish/bin e execute o script startserv para iniciar o Glassfish.
- 3 Verifique se o Glassfish está em execução, acessando <http://localhost:8080> através de um navegador.
- 4 Finalize o Glassfish executando o script stopserv, que está na mesma pasta do script startserv.
- 5 No Eclipse, abra a view “Servers” e clique com o botão direito no corpo dela. Escolha a opção “new” e configure o Glassfish.
- 6 Inicialize o Glassfish pela view “Servers” e verifique se ele está funcionando, acessando <http://localhost:8080>.
- 7 Finalize o Glassfish pela view “Servers”.



Aplicação Web Java

Para que uma aplicação web Java possa ser implantada em um Web Container, a estrutura de pastas precisa seguir algumas regras.

```
▷ K19-App/  
    ▷ WEB-INF/  
        ▷ classes/  
        ▷ lib/  
        ▷ web.xml
```

A pasta K19-App é a raiz da aplicação. Ela pode ter qualquer nome. A pasta WEB-INF deve ser criada dentro da pasta raiz. O conteúdo da pasta WEB-INF não pode ser acessado diretamente pelos usuários da aplicação. Por outro lado, os arquivos dentro da pasta raiz da aplicação mas fora da pasta WEB-INF podem ser acessados diretamente através de um navegador.

As pastas classes e lib devem ser criadas dentro da pasta WEB-INF. O código compilado da aplicação deve ser salvo na pasta classes. Os jar's das bibliotecas extras que serão utilizadas devem ser colocados na pasta lib. O arquivo web.xml contém configurações do Web Container e deve ser criado na pasta WEB-INF.

Em geral, as IDEs criam toda a estrutura de pastas exigidas pelos Web Containers. Então, na prática, não temos o trabalho de criar esses diretórios manualmente.



Mais Sobre

Consulte o artigo da K19 sobre criação de projetos web utilizando o Eclipse.

<http://www.k19.com.br/artigos/criando-um-dynamic-web-project/>



Exercícios de Fixação

- 8 No Eclipse, crie um projeto do tipo *Dynamic Web Project* chamado **K19-App**. Escolha “Glassfish” como opção para “Target runtime”. Na última tela de criação do projeto, selecione a opção **Generate web.xml deployment descriptor**.
- 9 Adicione o projeto K19-App no Glassfish através da view “Servers”. Clique com o botão direito do mouse no Glassfish e selecione “Add and Remove...”.
- 10 Inicialize o Glassfish através da view “Servers”. Clique com o botão direito do mouse sobre o Glassfish e escolha a opção “Start”.
- 11 Verifique o funcionamento da nossa aplicação acessando <http://localhost:8080/K19-App/> através de um navegador.



Processando requisições

Após implantar a nossa aplicação web Java em um Web Container, as requisições e respostas HTTP já estão sendo processadas pelo Web Container, que também já permite o acesso de múltiplos usuários à nossa aplicação.

Em seguida, devemos definir como o conteúdo das páginas da aplicação é gerado. Para isso, podemos criar uma **Servlet**.



Servlet

Para criar uma Servlet, podemos seguir os seguintes passos:

1. Criar uma classe.
2. Herdar da classe `javax.servlet.http.HttpServlet`.
3. Reescrever o método `service()`.
4. Utilizar a anotação `@WebServlet` para definir a url que será utilizada para acessar a Servlet. Essa anotação existe após a especificação Servlet 3.0. Antes, essa configuração era realizada através do arquivo `web.xml`.

```

1 @WebServlet("/OlaMundo")
2 public class OlaMundo extends HttpServlet{
3
4     @Override
5     protected void service(HttpServletRequest req, HttpServletResponse resp)
6         throws ServletException, IOException {
7         // Lógica para processar as regras de negócios e gerar conteúdo
8     }
9 }
```

Código Java 4.1: OlaMundo.java

O método `service()` é executado toda vez que uma requisição HTTP é realizada para a url definida na anotação `@WebServlet`. Esse método recebe dois parâmetros. O primeiro é a referência do objeto da classe `HttpServletRequest` que armazena todos os dados da requisição. O segundo parâmetro é a referência do objeto da classe `HttpServletResponse` que armazenará o conteúdo gerado pela Servlet.

Inserindo conteúdo na resposta

Para inserir conteúdo texto na resposta HTTP que será enviada para o navegador do usuário, devemos utilizar os métodos `getWriter()` e `println()`. Em geral, o conteúdo inserido na resposta HTTP é texto HTML. Veja o código abaixo.

```

1 @WebServlet("/OlaMundo")
2 public class OlaMundo extends HttpServlet {
3
4     @Override
5     protected void service(HttpServletRequest req, HttpServletResponse resp)
6         throws ServletException, IOException {
7         PrintWriter writer = resp.getWriter();
8         writer.println("<html><body><h1>Olá Mundo</h1></body></html>");
9     }
10 }
```

Código Java 4.2: OlaMundo.java

Exercícios de Fixação

12 Crie um pacote chamado `servlets` no projeto K19-App.

13 Crie uma classe chamada `OlaMundo` no pacote `servlets`.

```

1 @WebServlet("/OlaMundo")
2 public class OlaMundo extends HttpServlet {
3
4     @Override
5     protected void service(HttpServletRequest req, HttpServletResponse resp)
6         throws ServletException, IOException {
7         PrintWriter writer = resp.getWriter();
8         writer.println("<html><body><h1>Olá Mundo</h1></body></html>");
9     }
10 }
```

Código Java 4.3: OlaMundo.java

- 14 Verifique o funcionamento da Servlet acessando a url abaixo através de um navegador.

<http://localhost:8080/K19-App/OlaMundo>



Frameworks

Hoje em dia, é improvável que uma empresa decida começar um projeto utilizando diretamente Servlets, pois a produtividade seria pequena e a manutenção muito custosa. Vários frameworks foram criados para facilitar o desenvolvimento e a manutenção de aplicações web. Apesar de serem baseados em Servlets, esses frameworks oferecem diversos recursos adicionais para as aplicações. Eis uma lista dos principais frameworks para aplicações web Java:

- JSF
- Struts 1.x
- Struts 2.x
- Spring MVC

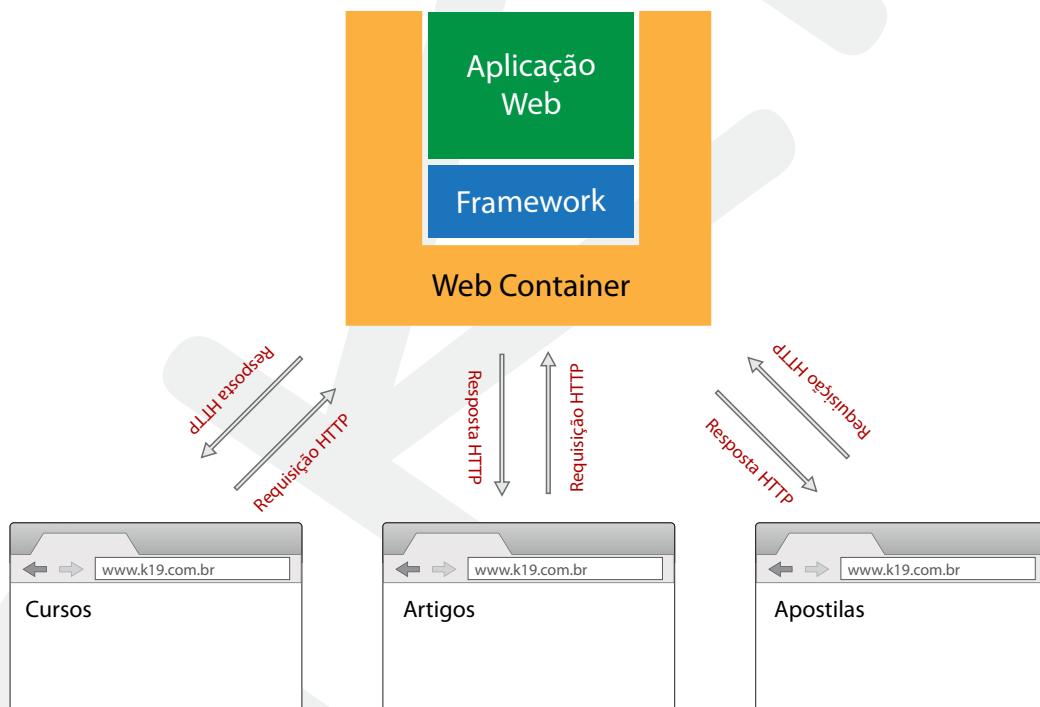


Figura 4.3: Framework para aplicações web

Nos próximos capítulos, mostraremos o funcionamento e explicaremos os conceitos relacionados ao framework Struts 2.



VISÃO GERAL DO STRUTS 2

Struts 2 é um framework para desenvolvimento de aplicações web em Java. A documentação desse framework pode ser obtida em <http://struts.apache.org/2.x/>. O Struts 2 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 na plataforma Java.

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 *Core J2EE Patterns: Best Practices and Design Strategies* (editora Prentice Hall, 2003, segunda edição) dos autores Deepak Alur, Dan Malks e John Crupi.



Bibliotecas

As bibliotecas do Struts podem ser obtidas na página de download desse framework (<http://struts.apache.org/download.cgi>). Neste treinamento, utilizaremos a versão 2.3.1.2 do Struts. Fazendo download do arquivo **struts-2.3.1.2-all.zip** e descompactando-o, você obterá as bibliotecas, o código fonte, alguns exemplos e a documentação do Struts 2.3.1.2.

A pasta struts-2.3.1.2/lib contém 82 arquivos .jar. Esses arquivos são as dependências do Struts 2.3.1.2. Desses arquivos, apenas 10 são obrigatórios. Os outros podem ser adicionados conforme a necessidade. Eis a lista dos 10 jars obrigatórios:

- commons-fileupload-1.2.2.jar
- commons-io-2.0.1.jar
- commons-lang-2.5.jar
- commons-logging-1.1.1.jar
- commons-logging-api-1.1.jar
- freemarker-2.3.18.jar
- ognl-3.0.4.jar
- struts2-core-2.3.1.2.jar
- xwork-core-2.3.1.2.jar
- javassist-3.11.0.GA.jar

Os arquivos .jar devem ser adicionadas na pasta WEB-INF/lib da aplicação.



Configurando uma aplicação Struts

Uma aplicação Struts deve respeitar a estrutura geral de uma aplicação web Java que foi descrita no capítulo anterior. Essa estrutura é definida pela especificação *Servlet* que está disponível em <http://www.jcp.org/en/jsr/detail?id=315>.

web.xml

O filtro do Struts deve ser configurado no arquivo WEB-INF/web.xml, indicando a classe que o implementa e o padrão de url que será associado a esse filtro.

Por exemplo, na configuração abaixo, todas as requisições que correspondem ao padrão de url “/*” serão processadas pelo filtro do Struts.

```
1 <filter>
2   <filter-name>struts2</filter-name>
3   <filter-class>
4     org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
5   </filter-class>
6 </filter>
7
8 <filter-mapping>
9   <filter-name>struts2</filter-name>
10  <url-pattern>/*</url-pattern>
11 </filter-mapping>
```

Código XML 5.1: web.xml

Consulte o endereço <http://struts.apache.org/2.x/docs/webxml.html> para mais informações.

struts.xml

Diversas configurações específicas do Struts podem ser realizadas através do arquivo struts.xml. Esse arquivo deve estar na raiz do class path da aplicação web. Utilizando a IDE eclipse, podemos defini-lo em uma source folder.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9     <action name="index">
10       <result>/Index</result>
11     </action>
12   </package>
13 </struts>
```

Código XML 5.2: struts.xml



Logging

Com intuito de encontrar mais facilmente os erros nas nossas aplicações, podemos ativar o logging do Struts. Para isso, devemos adicionar o arquivo log4j-1.2.14.jar na pasta WEB-INF/lib da aplicação. Esse arquivo pode ser obtido em <http://archive.apache.org/dist/logging/log4j/1.2.14/>.

Além disso, devemos definir as configurações de logging através do arquivo log4j.xml. Esse arquivo deve estar no class path da aplicação. Utilizando a IDE eclipse, podemos defini-lo em uma source folder.

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE log4j:configuration PUBLIC "-//log4j/log4j Configuration//EN" "log4j.dtd">
3
4 <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
5
6   <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
7     <layout class="org.apache.log4j.PatternLayout">
8       <param name="ConversionPattern" value="%d %-5p %c.%M:%L - %m%n"/>
9     </layout>
10    </appender>
11
12   <logger name="org.apache.struts2">
13     <level value="DEBUG" />
14   </logger>
15
16   <root>
17     <priority value="INFO"/>
18     <appender-ref ref="STDOUT" />
19   </root>
20 </log4j:configuration>
```

Código XML 5.3: log4j.xml



Actions

As actions são componentes fundamentais de uma aplicação Struts. Suas principais tarefas são:

1. Fornecer dados que serão exibidos nas telas.
2. Receber os dados enviados nas requisições.
3. Executar tarefas de acordo com as ações dos usuários.

Criando uma Action

Para definir uma action podemos criar uma classe Java herdando de **ActionSupport** e registrá-la no arquivo struts.xml. Veja o exemplo abaixo.

```
1 package br.com.k19.actions;
2
3 public class TesteAction extends ActionSupport {
4     ...
5 }
```

Código Java 5.1: TesteAction.java

```
1 ...
2 <action name="Teste" class="br.com.k19.actions.TesteAction">
3     ...
4 </action>
5 ...
```

Código XML 5.4: struts.xml

No registro de uma action, devemos definir o seu nome e a sua classe. O nome será utilizado para acessá-la através de um navegador.

Métodos de Ação

Podemos definir, na classe de uma action, o método que deverá ser executado toda vez que a action for acionada. Por padrão, esse método possui a seguinte estrutura:

```
1 package br.com.k19.actions;
2
3 public class TesteAction extends ActionSupport {
4     public String execute() throws Exception {
5         ...
6     }
7 }
```

Código Java 5.2: TesteAction.java

O resultado do método execute() pode ser utilizado para definir a próxima página a ser construída. Os resultados devem ser mapeados no arquivo struts.xml.

```
1 ...
2 <action name="Teste" class="br.com.k19.actions.TesteAction">
3     <result name="success">/Teste.jsp</result>
4 </action>
5 ...
```

Código XML 5.5: struts.xml

Podemos escolher um nome diferente para o método que será executado quando a action for acionada. Para isso, é necessário configurar o nome desse método no arquivo struts.xml.

```

1 package br.com.k19.actions;
2
3 public class TesteAction extends ActionSupport {
4     public String teste() throws Exception {
5         ...
6     }
7 }
```

Código Java 5.3: TesteAction.java

```

1 ...
2 <action name="Teste" method="teste" class="br.com.k19.actions.TesteAction">
3     <result name="success"/>/Teste.jsp</result>
4 </action>
5 ...
```

Código XML 5.6: struts.xml

Para evitar erros de digitação e padronizar as possíveis respostas de uma action, a interface Action define um conjunto de resultados padrão através de algumas constantes. A classe ActionSupport implementa a interface Action.

- SUCCESS
- NONE
- ERROR
- INPUT
- LOGIN

Podemos utilizar essas constantes nos métodos de ação.

```

1 package br.com.k19.actions;
2
3 public class TesteAction extends ActionSupport {
4     public String teste() throws Exception {
5         ...
6         return TesteAction.SUCCESS;
7     }
8 }
```

Código Java 5.4: TesteAction.java

Propriedades

Considere a seguinte action.

```

1 package br.com.k19.actions;
2
3 public class TesteAction extends ActionSupport {
4
5     private int numero;
6
7     public String execute() throws Exception {
8         ...
9     }
10 }
```

Código Java 5.5: TesteAction.java

Para acessar o valor do atributo numero em uma página, precisamos definir um método de leitura. Esse método deve seguir a convenção de nomenclatura do Java. Veja o exemplo abaixo:

```

1 package br.com.k19.actions;
2
3 public class TesteAction extends ActionSupport {
4
5     private int numero;
6
7     public String execute() throws Exception {
8         ...
9     }
10
11    public int getNumero() {
12        return numero;
13    }
14}
```

Código Java 5.6: TesteAction.java

Note que o nome do método começa com `get` e é seguido pelo nome do atributo com a primeira letra em caixa alta.

Para alterar o valor do atributo numero com valores obtidos através de uma página, precisamos definir um método de escrita.

```

1 package br.com.k19.actions;
2
3 public class TesteAction extends ActionSupport {
4
5     private int numero;
6
7     public String execute() throws Exception {
8         ...
9     }
10
11    public int getNumero() {
12        return numero;
13    }
14
15    public int setNumero(int numero) {
16        this.numero = numero;
17    }
18}
```

Código Java 5.7: TesteAction.java

O nome do método de escrita deve, necessariamente, começar com a palavra `set` e terminar com o nome do atributo com a primeira letra em caixa alta.

Com os métodos de acesso já implementados, podemos exibir o valor do atributo nas páginas de resultado da action TesteAction através da tag `<s:property>`. Para utilizar essa tag, devemos adicionar a diretiva `taglib` nas páginas. Veja o exemplo a seguir.

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3     <head>
4         <title>Teste</title>
```

```

5 </head>
6 <body>
7   <h1>Teste</h1>
8   <s:property value="numero"/>
9   </body>
10 </html>

```

Código JSP 5.1: Exibindo o valor do atributo numero

Para alterar o valor do atributo numero da action testeBean, podemos vinculá-lo, por exemplo, a uma caixa de texto em um formulário. Observe o código abaixo.

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3   <head>
4     <title>Formulário</title>
5   </head>
6   <body>
7     <s:form action="Teste">
8       <s:textfield name="numero"/>
9       <s:submit value="Enviar"/>
10    </s:form>
11  </body>
12 </html>

```

Código JSP 5.2: Alterando o valor do atributo numero



Importante

É importante destacar que o vínculo com uma propriedade de uma action dá-se por meio dos nomes dos métodos getters e setters, e não pelo nome do atributo.

Nos exemplos acima, se mantivéssemos o nome do atributo da action mas substituíssemos os nomes dos métodos getNumero() e setNumero() por getValor() e setValor(), respectivamente, então o valor do atributo value da tag <s:property> e valor do atributo name da tag <s:textfield> devem ser alterados.

```

1 <s:property value="valor"/>
1 <s:textfield name="valor"/>

```



Exemplo Prático

Com as configurações já realizadas, implementaremos uma aplicação que mostra o funcionamento básico do Struts. Essa aplicação deverá receber um texto do usuário e exibi-lo em letras maiúsculas.

Action

Vamos começar criando uma action para armazenar o texto enviado pelo usuário e a lógica para transformá-lo.

```

1 package br.com.k19.actions;

```

```

2 import com.opensymphony.xwork2.ActionSupport;
3
4 @SuppressWarnings("serial")
5 public class TextoAction extends ActionSupport {
6     private String texto;
7
8     @Override
9     public String execute() throws Exception {
10         this.texto = this.texto.toUpperCase();
11         return TextoAction.SUCCESS;
12     }
13
14     public String getTexto() {
15         return texto;
16     }
17
18     public void setTexto(String texto) {
19         this.texto = texto;
20     }
21 }
22

```

Código Java 5.8: TextoBean.java

A classe que implementa a action deve herdar de `ActionSupport`. O atributo `texto` armazenará o texto enviado pelo usuário e esse texto será modificado pelo método `execute()`. Esse método devolve uma string para indicar qual deve ser a próxima página a ser enviada para o usuário.

O Struts utilizará o método `setTexto()` para armazenar o texto enviado pelo usuário na action. Por outro lado, utilizará o método `getTexto()` para recuperar o texto e exibi-lo após a sua modificação.

No arquivo `struts.xml`, devemos registrar a action `TextoAction`.

```

1 ...
2 <action name="Texto" class="br.com.k19.actions.TextoAction">
3     <result name="success"/>/Texto.jsp</result>
4 </action>
5 ...

```

Código XML 5.7: struts.xml

Telas

Após a criação da action, podemos associá-la a um formulário que receberá o texto do usuário.

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3     <head>
4         <title>K19 Treinamentos</title>
5     </head>
6     <body>
7         <s:form action="Texto">
8             <s:textfield name="texto"/>
9             <s:submit value="Enviar"/>
10        </s:form>
11    </body>
12 </html>

```

Código JSP 5.5: Formulario.jsp

Observe nas linhas em destaque a ligação entre essa tela e a action. A caixa de entrada de texto foi associada à propriedade texto da action TextoAction. O formulário, por sua vez, foi associado à action TextoAction.

A tela de resultado deve apresentar o texto alterado.

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3   <head>
4     <title>K19 Treinamentos</title>
5   </head>
6   <body>
7     <h1>Texto em Caixa Alta</h1>
8     <s:property value="texto"/>
9   </body>
10 </html>
11 </html>
```

Código JSP 5.6: Texto.jsp

A tag <s:property> apresenta na tela o valor da propriedade texto da action TextoAction.



Exercícios de Fixação

Como exercício, desenvolva uma aplicação Struts que (i) receba um número inteiro do usuário, (ii) produza um número aleatório entre zero e o número recebido, e (iii) exiba esse número na tela do navegador do usuário.

- No Eclipse, crie um Dynamic Web Project chamado **K19-Visao-Geral**. Na primeira tela, devemos definir o nome do projeto e selecionar o target runtime.

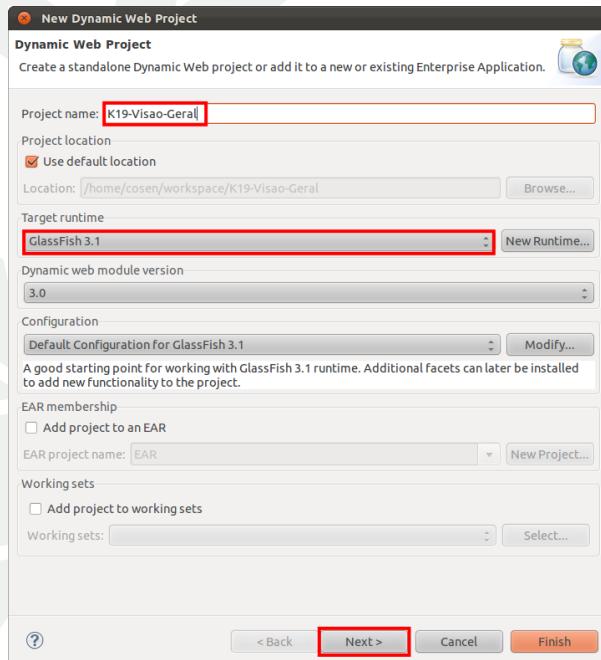


Figura 5.1: Criando um Dynamic Web Project

Na tela seguinte, apenas clique em “Next”.

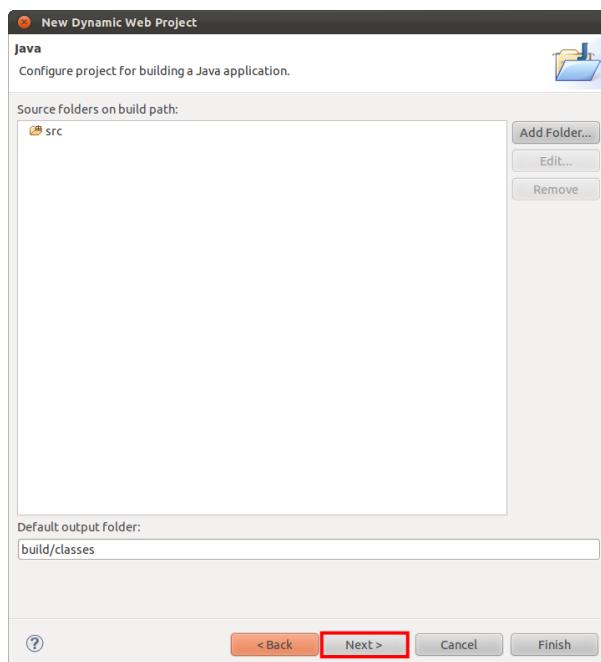


Figura 5.2: Criando um Dynamic Web Project

Na próxima tela, selecione a opção para gerar o arquivo web.xml e depois clique no botão “Finish”.

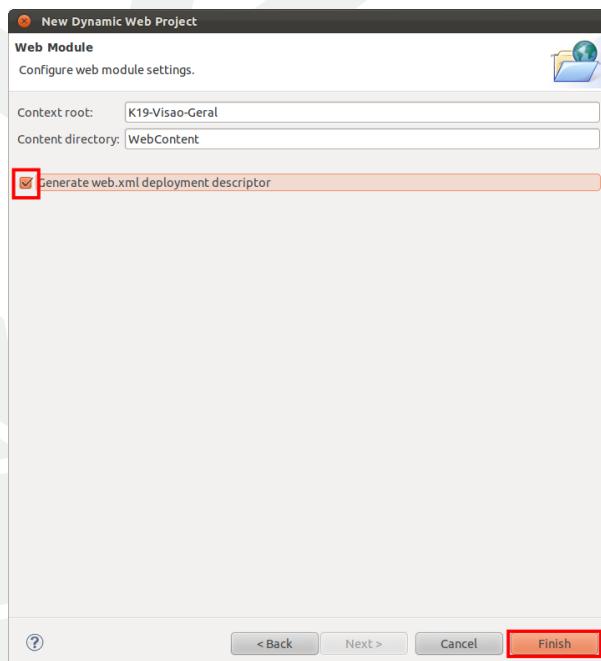


Figura 5.3: Criando um Dynamic Web Project

- 2** Adicione as bibliotecas do Struts na pasta WEB-INF/lib. Lembre-se que os arquivos das bibliotecas podem ser obtidos na página de download do Struts, <http://struts.apache.org/download.cgi>, baixando o arquivo struts-2.3.1.2-all.zip. Adicione apenas os jars obrigatórios na pasta WEB-INF/lib.

- commons-fileupload-1.2.2.jar
- commons-io-2.0.1.jar
- commons-lang-2.5.jar
- commons-logging-1.1.1.jar
- commons-logging-api-1.1.jar
- freemarker-2.3.18.jar
- ognl-3.0.4.jar
- struts2-core-2.3.1.2.jar
- xwork-core-2.3.1.2.jar
- javassist-3.11.0.GA.jar

- 3** Configure o filtro do Struts no arquivo web.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/←
4     javaee/web-app_2_5.xsd"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/←
6     javaee/web-app_3_0.xsd"
7   id="WebApp_ID" version="3.0">
8   <display-name>K19-Visao-Geral</display-name>
9   <welcome-file-list>
10    <welcome-file>index.html</welcome-file>
11    <welcome-file>index.htm</welcome-file>
12    <welcome-file>index.jsp</welcome-file>
13    <welcome-file>default.html</welcome-file>
14    <welcome-file>default.htm</welcome-file>
15    <welcome-file>default.jsp</welcome-file>
16   </welcome-file-list>
17   <filter>
18     <filter-name>struts2</filter-name>
19     <filter-class>
20       org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
21     </filter-class>
22   </filter>
23   <filter-mapping>
24     <filter-name>struts2</filter-name>
25     <url-pattern>/*</url-pattern>
26   </filter-mapping>
27 </web-app>
```

Código XML 5.8: web.xml

- 4** Adicione o arquivo de configuração do Struts, o struts.xml, na pasta src.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
```

```

3  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4  "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9
10  </package>
11 </struts>

```

Código XML 5.9: struts.xml

- 5 Adicione o arquivo de configuração para o logging do Struts, o log4j.xml, na pasta src.

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE log4j:configuration PUBLIC "-//log4j/log4j Configuration//EN" "log4j.dtd">
3
4 <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
5
6   <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
7     <layout class="org.apache.log4j.PatternLayout">
8       <param name="ConversionPattern" value="%d %-5p %c.%M:%L - %m%n"/>
9     </layout>
10    </appender>
11
12   <logger name="org.apache.struts2">
13     <level value="DEBUG" />
14   </logger>
15
16   <root>
17     <priority value="INFO"/>
18     <appender-ref ref="STDOUT" />
19   </root>
20 </log4j:configuration>

```

Código XML 5.10: log4j.xml

- 6 Crie um pacote chamado br.com.k19.actions. Nesse pacote, crie uma action para armazenar o número inteiro n enviado pelo usuário, gerar um número aleatório entre zero e n e armazena-lo numa propriedade.

```

1 package br.com.k19.actions;
2
3 import com.opensymphony.xwork2.ActionSupport;
4
5 @SuppressWarnings("serial")
6 public class NumeroAleatorioAction extends ActionSupport {
7
8   private int maximo;
9   private int numeroAleatorio;
10
11  public String execute() throws Exception {
12    this.numeroAleatorio = (int) (Math.random() * this.maximo);
13    return NumeroAleatorioAction.SUCCESS;
14  }
15
16  // GETTERS E SETTERS
17 }

```

Código Java 5.9: NumeroAleatorioAction.java

- 7 Registre a action NumeroAleatorioAction no struts.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9     <action name="NumeroAleatorio"
10       class="br.com.k19.actions.NumeroAleatorioAction">
11       <result name="success"/>/NumeroAleatorio.jsp</result>
12     </action>
13   </package>
14 </struts>

```

Código XML 5.11: struts.xml

- 8 Agora, na pasta WebContent, crie um formulário para que o usuário possa enviar o dado de entrada.

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2
3 <html>
4   <head>
5     <title>K19 Visão Geral</title>
6   </head>
7   <body>
8     <s:form action="NumeroAleatorio">
9       <s:label value="Número Máximo"/>
10      <s:textfield name="maximo"/>
11      <s:submit value="Gera número aleatório"/>
12    </s:form>
13  </body>
14 </html>

```

Código JSP 5.7: Formulario.jsp

- 9 Também na pasta WebContent, defina uma tela para exibir o número gerado aleatoriamente.

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3   <head>
4     <title>K19 Treinamentos</title>
5   </head>
6   <body>
7     <h1>Número Aleatório: <s:property value="numeroAleatorio"/></h1>
8   </body>
9 </html>

```

Código JSP 5.8: NumeroAleatorio.jsp

- 10 Adicione o projeto no GlassFish e teste-o, seguindo os passos abaixo.

1. Abra a aba “Servers”. Para isso, pressione “Ctrl+3”. Na janela que abrir, digite “Servers” e pressione “Enter”.

2. Na aba “Servers”, clique com o botão direito do mouse no GlassFish e selecione a opção “Add and Remove...”.
3. Selecione o projeto “K19-Visao-Geral” e clique em “Add >”. Em seguida, clique em “Finish”.
4. Inicialize o servidor. Para isso, clique mais uma vez no GlassFish com o botão direito e selecione “Start”.
5. Acesse a aplicação no endereço
<http://localhost:8080/K19-Visao-Geral/Formulario.jsp>.

INTEGRAÇÃO STRUTS E JPA

Como vimos nos primeiros capítulos, os SGDBs são utilizados para armazenar os dados manipulados pelas aplicações. Até agora, não discutimos como aplicações Struts podem armazenar dados em um SGDB através dos recursos definidos pelo JPA. Adicionaremos essa capacidade às aplicações Struts neste capítulo. Para isso, mostraremos uma maneira de integrar os recursos do Struts e do JPA.



Bibliotecas

Para utilizar os recursos do JPA em uma aplicação Struts, os jars do provedor JPA e do driver JDBC que serão utilizados devem estar no classpath da aplicação. No capítulo anterior, a aplicação Struts desenvolvida nos exercícios foi implantada no Glassfish 3.0.1 que é um servidor de aplicação Java EE 6.

Por padrão, a versão 3.0.1 do Glassfish possui os jars do provedor JPA EclipseLink. Dessa forma, as aplicações Struts implantadas nessa versão do Glassfish utilizarão o EclipseLink como implementação do JPA. Contudo, queremos utilizar o provedor JPA Hibernate. Podemos facilmente substituir os jars do EclipseLink pelos jars do Hibernate através da interface de administração do Glassfish.



Mais Sobre

Consulte o artigo da K19 sobre a substituição dos jars do provedor JPA EclipseLink pelos jars do provedor JPA Hibernate.

<http://www.k19.com.br/artigos/configurando-hibernate-no-glassfish-3-1/>

Utilizaremos o MySQL Server como SGDB. Dessa forma, devemos adicionar o driver JDBC do MySQL Server no classpath das aplicações Struts. O Glassfish 3.0.1 não possui os jars desse driver JDBC. Contudo, podemos adicioná-los manualmente. Para isso, basta acrescentar os jars do driver JDBC do MySQL em uma pasta apropriada do Glassfish. Nos exercícios deste capítulo, mostraremos como realizar tal tarefa.



Configuração

Como vimos no Capítulo 3, devemos configurar as unidades de persistência utilizadas através do arquivo `persistence.xml` da pasta META-INF do classpath da aplicação.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0"
3   xmlns="http://java.sun.com/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

5   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/←
6     ns/persistence/persistence_2_0.xsd">
7   <persistence-unit name="K19-PU" transaction-type="RESOURCE_LOCAL">
8     <provider>org.hibernate.ejb.HibernatePersistence</provider>
9     <properties>
10       <property name="hibernate.dialect" value="org.hibernate.dialect.←
11         MySQL5InnoDBialect"/>
12       <property name="hibernate.hbm2ddl.auto" value="create"/>
13       <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
14       <property name="javax.persistence.jdbc.user" value="root"/>
15       <property name="javax.persistence.jdbc.password" value="root"/>
16       <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/←
17         K19_DB"/>
18     </properties>
19   </persistence-unit>
20 </persistence>
21
22
23

```

Código XML 6.1: persistence.xml

Mapeamento

Também devemos definir o mapeamento das entidades. No Capítulo 3, vimos como utilizar as anotações do JPA para estabelecer esse mapeamento.

```

1 @Entity
2 public class Produto {
3   @Id @GeneratedValue
4   private Long id;
5
6   private String nome;
7
8   private Double preco;
9
10  // GETTERS e SETTERS
11 }

```

Código Java 6.1: Produto.java

Inicialização e Finalização

As unidades de persistência devem ser inicializadas antes de serem utilizadas, e finalizadas quando não forem mais necessárias. A inicialização e a finalização de uma unidade de persistência devem ser realizadas apenas uma vez durante a execução da aplicação.

Para implementar essa característica em aplicações web Java, podemos utilizar um filtro. Os filtros de uma aplicação web Java são inicializados automaticamente depois que a aplicação é implantada no Web Container e antes da primeira requisição HTTP. Além disso, eles são finalizados ao término da execução da aplicação.

Para adicionar um filtro em uma aplicação web Java, é necessário criar uma classe que implemente a interface `javax.servlet.Filter`.

```

1 public class JPAFilter implements Filter {
2
3     private EntityManagerFactory factory;
4
5     @Override
6     public void init(FilterConfig filterConfig) throws ServletException {
7         this.factory = Persistence.createEntityManagerFactory("K19-PU");
8     }
9
10    @Override
11    public void destroy() {
12        this.factory.close();
13    }
14
15    @Override
16    public void doFilter(ServletRequest request, ServletResponse response,
17        FilterChain chain) throws IOException, ServletException {
18
19        // por enquanto vazio
20    }
21 }
```

Código Java 6.2: JPAFilter.java

Um filtro pode ser registrado no Web Container através do arquivo web.xml. Nesse registro, podemos definir a ordem na qual os filtros devem ser executados. No exemplo abaixo, o filtro definido pela classe JPAFilter executará antes do filtro do Struts. Além disso, ele está mapeado para o mesmo padrão de url.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/←
4     javaee/web-app_2_5.xsd"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/←
6     javaee/web-app_3_0.xsd"
7   id="WebApp_ID" version="3.0">
8
9   <display-name>K19-Visao-Geral-Struts</display-name>
10  <welcome-file-list>
11    <welcome-file>index.html</welcome-file>
12    <welcome-file>index.htm</welcome-file>
13    <welcome-file>index.jsp</welcome-file>
14    <welcome-file>default.html</welcome-file>
15    <welcome-file>default.htm</welcome-file>
16    <welcome-file>default.jsp</welcome-file>
17  </welcome-file-list>
18
19  <filter>
20    <filter-name>JPAFilter</filter-name>
21    <filter-class>br.com.k19.filtros.JPAFilter</filter-class>
22  </filter>
23
24  <filter-mapping>
25    <filter-name>JPAFilter</filter-name>
26    <url-pattern>/*</url-pattern>
27  </filter-mapping>
28
29  <filter>
30    <filter-name>struts2</filter-name>
31    <filter-class>
32      org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-←
33      class>
34  </filter>
35
36  <filter-mapping>
37    <filter-name>struts2</filter-name>
38    <url-pattern>/*</url-pattern>
```

```
36 </filter-mapping>
37 </web-app>
```

Código XML 6.2: web.xml

O método **init()** é chamado automaticamente na inicialização do filtro. No exemplo acima, esse método inicializa a unidade de persistência *K19-PU*. O método **destroy()** é chamado automaticamente para desativar o filtro no encerramento da aplicação. No exemplo acima, finalizamos a unidade de persistência *K19-PU*.



Transações

Como vimos no Capítulo 3, para atualizar as informações armazenadas no SGDB de acordo com os dados da memória da aplicação, devemos abrir uma transação e confirmá-la através do método **commit()**.

O filtro criado anteriormente para controlar a inicialização e finalização das unidades de persistência pode também gerenciar a abertura e a confirmação das transações da aplicação. Para isso, utilizaremos o método **doFilter()** desse filtro.

```
1 public class JPAFilter implements Filter {
2
3     private EntityManagerFactory factory;
4
5     @Override
6     public void init(FilterConfig filterConfig) throws ServletException {
7         this.factory = Persistence.createEntityManagerFactory("K19-PU");
8     }
9
10    @Override
11    public void destroy() {
12        this.factory.close();
13    }
14
15    @Override
16    public void doFilter(ServletRequest request, ServletResponse response,
17        FilterChain chain) throws IOException, ServletException {
18
19        // CHEGADA
20        EntityManager manager = this.factory.createEntityManager();
21        request.setAttribute("EntityManager", manager);
22        entityManager.getTransaction().begin();
23        // CHEGADA
24
25        // FILTRO DO STRUTS
26        chain.doFilter(request, response);
27        // FILTRO DO STRUTS
28
29        // SAÍDA
30        try {
31            entityManager.getTransaction().commit();
32        } catch (Exception e) {
33            entityManager.getTransaction().rollback();
34        } finally {
35            entityManager.close();
36        }
37        // SAÍDA
38    }
39}
```

Código Java 6.3: JPAFilter.java

No exemplo acima, o método `doFilter()` é chamado quando uma requisição com o padrão de url “`/*`” é realizada. Antes de repassar a requisição para o filtro do Struts, o método `doFilter()` cria um `EntityManager`, armazena-o na requisição e abre uma transação. Depois que o filtro do Struts terminar o seu processamento, o método `doFilter()` tenta confirmar a transação através do método `commit()`. Se um erro ocorrer nessa tentativa, o método `rollback()` é chamado para cancelar a transação.



Recuperando o EntityManager da Requisição

O `EntityManager` armazenado dentro da requisição pelo filtro pode ser recuperado a qualquer momento durante o processamento da requisição. Veja o código abaixo.

```
1 HttpServletRequest request = ServletActionContext.getRequest();
2 EntityManager manager = (EntityManager) request.getAttribute("EntityManager");
```

Código Java 6.4: Recuperando o EntityManager da requisição

O `EntityManager` será utilizado pela aplicação para realizar as operações de persistência.



Exercícios de Fixação

- 1 Entre na pasta K19-Arquivos/MySQL-Connector-JDBC da Área de Trabalho e copie o arquivo **mysql-connector-java-5.1.13-bin.jar** para pasta glassfishv3/glassfish/lib também da sua Área de Trabalho. OBS: O Glassfish deve ser reiniciado para reconhecer o driver JDBC do MySQL.
- 2 Crie um projeto do tipo *Dynamic Web Project* chamado **K19-Integracao-Struts-JPA** seguindo os passos vistos nos exercícios 1 ao 5 do Capítulo 5.
- 3 Adicione uma pasta chamada META-INF na pasta src do projeto K19-Integracao-Struts-JPA.
- 4 Configure o JPA adicionando o arquivo `persistence.xml` na pasta src/META-INF.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0">
3   xmlns="http://java.sun.com/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/←
6   ns/persistence/persistence_2_0.xsd">
7     <persistence-unit name="K19-PU" transaction-type="RESOURCE_LOCAL">
8       <provider>org.hibernate.ejb.HibernatePersistence</provider>
9       <properties>
10         <property name="hibernate.dialect" value="org.hibernate.dialect.←
11           MySQL5InnoDBDialect"/>
12         <property name="hibernate.hbm2ddl.auto" value="update"/>
13         <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
14     </persistence-unit>
15 </persistence>
```

```

16     <property name="javax.persistence.jdbc.user" value="root"/>
17
18     <property name="javax.persistence.jdbc.password" value="root"/>
19
20     <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/←
21         K19_DB"/>
22   </properties>
23 </persistence-unit>
24 </persistence>

```

Código XML 6.3: persistence.xml

- 5** Abra um terminal; entre no cliente do MySQL Server; apague se existir a base de dados K19-DB; e crie uma base de dados nova chamada K19-DB.

```

k19@k19-11:~/rafael$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 36
Server version: 5.1.58-1ubuntu1 (Ubuntu)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> DROP DATABASE IF EXISTS K19_DB;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE DATABASE K19_DB;
Query OK, 1 row affected (0.02 sec)

```

- 6** Crie um pacote chamado br.com.k19.filters na pasta src do projeto K19-Integracao-Struts-JPA.

- 7** No pacote br.com.k19.filters, crie uma classe chamada JPAFilter com o seguinte conteúdo:

```

1 package br.com.k19.filters;
2
3 import java.io.IOException;
4 import javax.persistence.*;
5 import javax.servlet.*;
6
7 public class JPAFilter implements Filter {
8
9     private EntityManagerFactory factory;
10
11    @Override
12    public void init(FilterConfig filterConfig) throws ServletException {
13        this.factory = Persistence.createEntityManagerFactory("K19-PU");
14    }
15
16    @Override
17    public void destroy() {
18        this.factory.close();
19    }
20
21    @Override
22    public void doFilter(ServletRequest request, ServletResponse response,
23                         FilterChain chain) throws IOException, ServletException {
24
25        // CHEGADA
26        EntityManager manager = this.factory.createEntityManager();

```

```

27     request.setAttribute("EntityManager", manager);
28     manager.getTransaction().begin();
29     // CHEGADA
30
31     // FILTRO DO STRUTS
32     chain.doFilter(request, response);
33     // FILTRO DO STRUTS
34
35     // SAÍDA
36     try {
37         manager.getTransaction().commit();
38     } catch (Exception e) {
39         manager.getTransaction().rollback();
40     } finally {
41         manager.close();
42     }
43     // SAÍDA
44 }
45 }
```

Código Java 6.5: JPAFilter.java

8 Registre o JPAFilter alterando o arquivo web.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/←
4     javaee/web-app_2_5.xsd"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/←
6     javaee/web-app_3_0.xsd"
7   id="WebApp_ID" version="3.0">
8
9   <display-name>K19-Integracao-Struts-JPA</display-name>
10  <welcome-file-list>
11    <welcome-file>index.html</welcome-file>
12    <welcome-file>index.htm</welcome-file>
13    <welcome-file>index.jsp</welcome-file>
14    <welcome-file>default.html</welcome-file>
15    <welcome-file>default.htm</welcome-file>
16    <welcome-file>default.jsp</welcome-file>
17  </welcome-file-list>
18
19  <filter>
20    <filter-name>JPAFilter</filter-name>
21    <filter-class>br.com.k19.filters.JPAFilter</filter-class>
22  </filter>
23  <filter-mapping>
24    <filter-name>JPAFilter</filter-name>
25    <url-pattern>/*</url-pattern>
26  </filter-mapping>
27
28  <filter>
29    <filter-name>struts2</filter-name>
30    <filter-class>org.apache.struts2.dispatcher.ng.filter.←
31      StrutsPrepareAndExecuteFilter</filter-class>
32  </filter>
33
34  <filter-mapping>
35    <filter-name>struts2</filter-name>
36    <url-pattern>/*</url-pattern>
37  </filter-mapping>
38
39 </web-app>
```

Código XML 6.4: web.xml

- 9** Crie um pacote chamado br.com.k19.model na pasta src do projeto K19-Integracao-Struts-JPA.

- 10** No pacote br.com.k19.model, crie uma classe chamada Carro com o seguinte conteúdo:

```

1 package br.com.k19.model;
2
3 // IMPORTS
4
5 @Entity
6 public class Carro {
7
8     @Id @GeneratedValue
9     private Long id;
10    private String marca;
11    private String modelo;
12
13    // GETTERS E SETTERS
14 }
```

Código Java 6.6: Carro.java

- 11** Também no pacote br.com.k19.model, crie uma classe chamada CarroRepository com o seguinte conteúdo:

```

1 package br.com.k19.model;
2
3 import java.util.List;
4 import javax.persistence.EntityManager;
5 import javax.persistence.Query;
6
7 public class CarroRepository {
8
9     private EntityManager manager;
10
11    public CarroRepository(EntityManager manager) {
12        this.manager = manager;
13    }
14
15    public void adiciona(Carro carro) {
16        this.manager.persist(carro);
17    }
18
19    public List<Carro> buscaTodos() {
20        Query query = this.manager.createQuery("select x from Carro x");
21        return query.getResultList();
22    }
23 }
```

Código Java 6.7: CarroRepository.java

- 12** Crie um pacote chamado br.com.k19.actions na pasta src do projeto K19-Integracao-Struts-JPA.

- 13** No pacote br.com.k19.actions, crie uma classe chamada AdicionaCarroAction com o seguinte código:

```

1 package br.com.k19.actions;
2
```

```

3 import java.util.List;
4 import javax.persistence.EntityManager;
5 import javax.servlet.http.HttpServletRequest;
6 import org.apache.struts2.ServletActionContext;
7 import br.com.k19.model.Carro;
8 import br.com.k19.model.CarroRepository;
9 import com.opensymphony.xwork2.ActionSupport;
10
11 @SuppressWarnings("serial")
12 public class AdicionaCarroAction extends ActionSupport {
13
14     private Carro carro = new Carro();
15
16     public String execute() throws Exception {
17         EntityManager manager = this.getEntityManager();
18         CarroRepository repository = new CarroRepository(manager);
19
20         repository.adiciona(this.carro);
21         this.carro = new Carro();
22         return AdicionaCarroAction.SUCCESS;
23     }
24
25     public List<Carro> getCarros() {
26         EntityManager manager = this.getEntityManager();
27         CarroRepository repository = new CarroRepository(manager);
28         return repository.buscaTodos();
29     }
30
31     private EntityManager getEntityManager() {
32         HttpServletRequest request = ServletActionContext.getRequest();
33         EntityManager manager = (EntityManager)request.getAttribute("EntityManager");
34
35         return manager;
36     }
37
38     public Carro getCarro() {
39         return carro;
40     }
41
42     public void setCarro(Carro carro) {
43         this.carro = carro;
44     }
45 }
```

Código Java 6.8: AdicionaCarroAction.java

- 14** Registre a action AdicionaCarroAction alterando o arquivo struts.xml no pacote WebContent/META-INF.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9     <action name="AdicionaCarro" class="br.com.k19.actions.AdicionaCarroAction">
10       <result name="success"/>/Lista.jsp</result>
11     </action>
12   </package>
13 </struts>
```

Código XML 6.5: struts.xml

- 15** No diretório WebContent, crie um arquivo JSP chamado `Formulario.jsp` com o seguinte conteúdo:

```

1 <%@ taglib prefix="s" uri="/struts-tags"%>
2 <html>
3   <head>
4     <title>K19 Treinamentos</title>
5   </head>
6   <body>
7     <s:form action="AdicionaCarro">
8       <s:label value="Marca: "/>
9       <s:textfield name="carro.marca" />
10      <s:label value="Modelo: "/>
11      <s:textfield name="carro.modelo" />
12      <s:submit value="Enviar" />
13    </s:form>
14  </body>
15 </html>
```

Código JSP 6.1: Formulario.jsp

- 16** Crie o arquivo de resposta, o `Lista.jsp`, na pasta WebContent, para apresentar os carros adicionados.

```

1 <%@ taglib prefix="s" uri="/struts-tags"%>
2 <html>
3   <head>
4     <title>K19 Treinamentos</title>
5   </head>
6   <body>
7     <ul>
8       <s:iterator value="carros">
9         <li><s:property value="marca"/> - <s:property value="modelo"/></li>
10      </s:iterator>
11    </ul>
12  </body>
13 </html>
```

Código JSP 6.2: Lista.jsp

Note que utilizamos a tag `<s:iterator>` para percorrer todos os elementos da lista de carros. No corpo dessa tag, podemos acessar as propriedades de cada elemento simplesmente com os nomes dessas propriedades. Por exemplo, para acessar a propriedade `marca`, usamos o identificador `"marca"` com a tag `<s:property>`.

- 17** Acesse a aplicação no endereço:

`http://localhost:8080/K19-Integracao-Struts-JPA/Formulario.jsp`

Adicione alguns carros e verifique se eles foram adicionados no SGDB.

AUTENTICAÇÃO

Neste capítulo, apresentaremos uma maneira de implementar o processo de autenticação dos usuários de uma aplicação Struts.



Exercícios de Fixação

- 1 Crie um projeto do tipo *Dynamic Web Project* chamado **K19-Autenticacao** seguindo os passos vistos nos exercícios do 1 ao 5 no Capítulo 5.
- 2 Por simplicidade, utilizaremos um atributo estático de uma action para armazenar os usuários da aplicação e suas respectivas senhas. A implementação que será apresentada a seguir pode ser alterada para que esses dados sejam armazenadas em um arquivo ou em um banco de dados.



Importante

Por motivos de segurança, as senhas dos usuários não devem ser armazenadas literalmente. Ao invés disso, as senhas dos usuários devem passar por um processo de transformação (criptografia) antes de serem armazenadas.

Quando um usuário tenta logar no sistema, ele digita o seu nome de usuário e sua senha. Para garantir que o usuário tenha acesso ao sistema, precisamos verificar se o nome de usuário digitado está cadastrado no sistema e se sua senha está correta. Como nós não armazenamos a senha do usuário, o que fazemos é aplicar a mesma transformação feita anteriormente e comparar o valor obtido com aquele armazenado no servidor. Se esses valores forem iguais, então permitimos que o usuário acesse o sistema. Caso contrário, o acesso ao sistema é negado.

Crie uma classe chamada `LoginAction` em um pacote chamado `br.com.k19.actions` no projeto `K19-Autenticacao` com o seguinte conteúdo:

```
1 package br.com.k19.actions;
2
3 import java.util.HashMap;
4 import java.util.Map;
5 import com.opensymphony.xwork2.ActionSupport;
6
7 @SuppressWarnings("serial")
8 public class LoginAction extends ActionSupport {
9     private static Map<String, String> mapa = new HashMap<String, String>();
10 }
```

Código Java 7.1: `LoginAction.java`

- 3 Acrescente alguns usuários e suas respectivas senhas no atributo mapa.

```

1 package br.com.k19.actions;
2
3 import java.util.HashMap;
4 import java.util.Map;
5 import com.opensymphony.xwork2.ActionSupport;
6
7 @SuppressWarnings("serial")
8 public class LoginAction extends ActionSupport {
9     private static Map<String, String> mapa = new HashMap<String, String>();
10
11     static {
12         LoginAction.mapa.put("k19", "k19");
13         LoginAction.mapa.put("jonas.hirata", "jonas.hirata");
14         LoginAction.mapa.put("marcelo.martins", "marcelo.martins");
15         LoginAction.mapa.put("rafael.cosentino", "rafael.cosentino");
16     }
17 }
```

Código Java 7.2: LoginAction.java

- 4 Crie propriedades para armazenar os dados enviados através do formulário de identificação e um método para implementar o processo de autenticação.

```

1 package br.com.k19.actions;
2
3 import java.util.HashMap;
4 import java.util.Map;
5 import javax.servlet.http.HttpSession;
6 import org.apache.struts2.ServletActionContext;
7 import com.opensymphony.xwork2.ActionSupport;
8
9 @SuppressWarnings("serial")
10 public class LoginAction extends ActionSupport {
11     private static Map<String, String> mapa = new HashMap<String, String>();
12
13     private String usuario;
14
15     private String senha;
16
17     private String mensagem;
18
19     static {
20         LoginAction.mapa.put("k19", "k19");
21         LoginAction.mapa.put("jonas.hirata", "jonas.hirata");
22         LoginAction.mapa.put("marcelo.martins", "marcelo.martins");
23         LoginAction.mapa.put("rafael.cosentino", "rafael.cosentino");
24     }
25
26     public String execute() throws Exception {
27         if (LoginAction.mapa.containsKey(this.usuario)
28             && LoginAction.mapa.get(this.usuario).equals(this.senha)) {
29
30             HttpSession session =
31                 ServletActionContext.getRequest().getSession( true);
32             session.setAttribute("usuario", this.usuario);
33
34             this.mensagem = "Autenticação realizada com sucesso.";
35             return LoginAction.SUCCESS;
36         } else {
37             this.mensagem = "Usuário e/ou senha incorretos.";
38             return LoginAction.INPUT;
39         }
40     }
41 }
```

```
42 // GETTERS E SETTERS
43 }
```

Código Java 7.3: LoginAction.java

- 5 Crie uma classe chamada LogoutAction no pacote br.com.k19.actions no projeto K19-Autenticacao com o seguinte conteúdo:

```
1 package br.com.k19.actions;
2
3 import javax.servlet.http.HttpSession;
4 import org.apache.struts2.ServletActionContext;
5 import com.opensymphony.xwork2.ActionSupport;
6
7 @SuppressWarnings("serial")
8 public class LogoutAction extends ActionSupport {
9
10    private String mensagem;
11
12    public String execute() throws Exception {
13
14        HttpSession session = ServletActionContext.getRequest()
15            .getSession(true);
16        session.removeAttribute("usuario");
17
18        this.mensagem = "Até logo!";
19
20        return LogoutAction.SUCCESS;
21    }
22
23    // GETTERS E SETTERS
24 }
```

Código Java 7.4: LogoutAction.java

- 6 Registre as duas actions criadas anteriormente alterando o arquivo struts.xml.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9     <action name="Login" class="br.com.k19.actions.LoginAction">
10       <result name="success">/Home.jsp</result>
11       <result name="input">/Login.jsp</result>
12     </action>
13     <action name="Logout" class="br.com.k19.actions.LogoutAction">
14       <result name="success">/Login.jsp</result>
15     </action>
16   </package>
17 </struts>
```

Código XML 7.1: struts.xml

- 7 Para testar, crie a tela do formulário de autenticação e a tela principal da aplicação.

```
1 <%@ taglib prefix="s" uri="/struts-tags"%>
2 <html>
```

```

3 <head>
4   <title>K19 Treinamentos</title>
5 </head>
6 <body>
7   <div style="color: red">
8     <s:property value="mensagem"/>
9   </div>
10
11  <s:form action="Login">
12    <s:label value="Usuário: "/>
13    <s:textfield name="usuario" />
14    <s:label value="Senha: "/>
15    <s:password name="senha" />
16    <s:submit value="Enviar" />
17  </s:form>
18 </body>
19 </html>

```

Código JSP 7.1: Login.jsp

```

1 <%@ taglib prefix="s" uri="/struts-tags"%>
2 <html>
3   <head>
4     <title>K19 Treinamentos</title>
5   </head>
6   <body>
7     <div style="color: red">
8       <s:property value="mensagem"/>
9     </div>
10
11    <s:url action="Logout" var="url"/>
12    <s:a href="#">Logout</s:a>
13  </body>
14 </html>

```

Código JSP 7.2: Home.jsp

- 8 Teste a aplicação acessando <http://localhost:8080/K19-Autenticacao/Login.jsp>

- 9 Após fazer logout, tente acessar a página <http://localhost:8080/K19-Autenticacao/Home.jsp>. Note que você é capaz de acessá-la. No entanto, somente usuários autenticados podem acessar a página principal da aplicação. Para controlar o acesso às páginas da aplicação, implemente um filtro para interceptar todas as requisições HTTP direcionadas ao filtro do Struts.

Crie uma classe chamada ControleDeAcesso em um pacote chamado br.com.k19.filters no projeto K19-Autenticacao com o seguinte conteúdo:

```

1 package br.com.k19.filters;
2
3 import java.io.IOException;
4
5 import javax.servlet.Filter;
6 import javax.servlet.FilterChain;
7 import javax.servlet.FilterConfig;
8 import javax.servlet.ServletException;
9 import javax.servlet.ServletRequest;
10 import javax.servlet.ServletResponse;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13 import javax.servlet.http.HttpSession;
14
15 public class ControleDeAcesso implements Filter {

```

```

16
17     @Override
18     public void doFilter(ServletRequest request, ServletResponse response,
19             FilterChain chain) throws IOException, ServletException {
20
21         HttpServletRequest req = (HttpServletRequest) request;
22         HttpSession session = req.getSession();
23
24         if (session.getAttribute("usuario") != null
25             || req.getRequestURI().endsWith("Login.jsp")
26             || req.getRequestURI().endsWith("Login.action")) {
27             chain.doFilter(request, response);
28         } else {
29             HttpServletResponse res = (HttpServletResponse) response;
30             res.sendRedirect("Login.jsp");
31         }
32     }
33
34     @Override
35     public void init(FilterConfig filterConfig) throws ServletException {
36
37     }
38
39     @Override
40     public void destroy() {
41
42     }
43 }
```

Código Java 7.5: ControleDeAcesso.java

- 10** Registre o filtro ControleDeAcesso alterando o arquivo web.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/←
4     javaee/web-app_2_5.xsd"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/←
6     javaee/web-app_3_0.xsd"
7   id="WebApp_ID" version="3.0">
8
9   <display-name>K19-Visao-Geral-Struts</display-name>
10  <welcome-file-list>
11    <welcome-file>index.html</welcome-file>
12    <welcome-file>index.htm</welcome-file>
13    <welcome-file>index.jsp</welcome-file>
14    <welcome-file>default.html</welcome-file>
15    <welcome-file>default.htm</welcome-file>
16    <welcome-file>default.jsp</welcome-file>
17
18    <filter>
19      <filter-name>Controle De Acesso</filter-name>
20      <filter-class>br.com.k19.filters.ControleDeAcesso</filter-class>
21    </filter>
22
23    <filter-mapping>
24      <filter-name>Controle De Acesso</filter-name>
25      <url-pattern>/*</url-pattern>
26    </filter-mapping>
27
28    <filter>
29      <filter-name>struts2</filter-name>
30      <filter-class>org.apache.struts2.dispatcher.ng.filter.←
31          StrutsPrepareAndExecuteFilter</filter-class>
```

```
32 <filter-mapping>
33   <filter-name>struts2</filter-name>
34   <url-pattern>/*</url-pattern>
35 </filter-mapping>
36 </web-app>
```

Código XML 7.2: web.xml

- 11 Reinicie a aplicação e tente acessar diretamente a página principal da aplicação acessando <http://localhost:8080/K19-Autenticacao/Home.jsp>. Observe que a aplicação redireciona o navegador para a página do formulário de autenticação.

PÁGINAS DE ERRO

Por padrão, quando determinados erros ocorrem no processamento de uma requisição, páginas com informações técnicas sobre o problema que ocorreu são geradas e enviadas para os usuários. Na fase de desenvolvimento, essas páginas são úteis para os desenvolvedores. Por outro lado, na fase de produção, essas páginas podem confundir os usuários da aplicação e revelar a estrutura do sistema, expondo possíveis falhas de segurança.

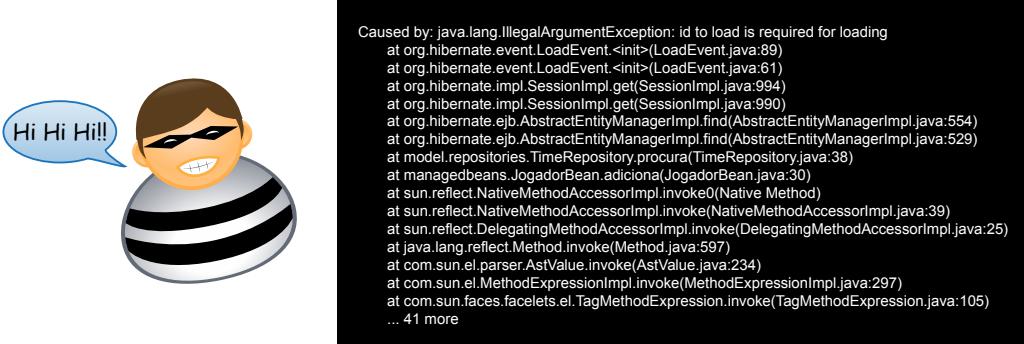


Figura 8.1: Expondo possíveis falhas de segurança

Neste capítulo, apresentaremos uma maneira de personalizar as páginas de erro da aplicação.



Exercícios de Fixação

- Crie um projeto do tipo *Dynamic Web Project* chamado **K19-Paginas-De-Erro** seguindo os passos vistos nos exercícios 1 ao 5 do Capítulo 5.
- Criaremos uma página de erro padrão. Adicione na pasta *WebContent* um arquivo JSP com o seguinte conteúdo.

```
1 <html>
2 <head>
3   <title>K19 Treinamentos</title>
4 </head>
5
6 <body>
7   <h1>Ocorreu um erro interno no sistema.</h1>
8   <h3>Tente novamente em alguns instantes.</h3>
9 </body>
10 </html>
```

Código JSP 8.1: Pagina-De-Erro.jsp

- 3 Criaremos uma página com um botão que sempre produzirá um erro ao ser clicado. Adicione na pasta WebContent um arquivo JSP com o seguinte conteúdo.

```

1 <%@ taglib prefix="s" uri="/struts-tags"%>
2 <html>
3   <head>
4     <title>K19 Treinamentos</title>
5   </head>
6   <body>
7     <s:form action="Erro">
8       <s:submit value="Gera Erro" />
9     </s:form>
10   </body>
11 </html>
```

Código JSP 8.2: Erro.jsp

- 4 Crie uma action que provoque propositalmente um erro. Adicione em um pacote chamado br.com.k19.actions uma classe com o seguinte código:

```

1 package br.com.k19.actions;
2
3 import com.opensymphony.xwork2.ActionSupport;
4
5 @SuppressWarnings("serial")
6 public class ErroAction extends ActionSupport {
7
8   @Override
9   public String execute() throws Exception {
10     System.out.println(10/0);
11     return ErroAction.SUCCESS;
12   }
13 }
```

Código Java 8.1: ErroAction.java

- 5 Registre a action ErroAction alterando o arquivo struts.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9     <action name="Erro" class="br.com.k19.actions.ErroAction">
10       <result name="success">/Erro.jsp</result>
11     </action>
12   </package>
13 </struts>
```

Código XML 8.1: struts.xml

- 6 Teste a aplicação acessando o seguinte endereço:

<http://localhost:8080/K19-Paginas-De-Erro/Erro.jsp>

- 7 Configure o Struts para direcionar todas as exceptions para a página de erro padrão. Altere o código do arquivo struts.xml.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9     <global-results>
10       <result name="error"/>/Pagina-De-Erro.jsp</result>
11     </global-results>
12
13   <global-exception-mappings>
14     <exception-mapping result="error" exception="java.lang.Exception"/>
15   </global-exception-mappings>
16
17   <action name="Erro" class="br.com.k19.actions.ErroAction">
18     <result name="success"/>/Erro.jsp</result>
19   </action>
20 </package>
21 </struts>
```

Código XML 8.2: struts.xml

A tag <global-exception-mappings> acima define que qualquer action que lançar uma java.lang.Exception produzirá o resultado ‘‘error’’. A tag <global-results> acima determina que toda vez que uma action gerar o resultado ‘‘error’’, a página a ser apresentada ao usuário será /Pagina-De-Erro.jsp.

- 8 Teste a aplicação novamente acessando o seguinte endereço:

<http://localhost:8080/K19-Paginas-De-Erro/Erro.jsp>



CRUD

Neste capítulo, criaremos uma pequena aplicação CRUD com Struts para exemplificar o funcionamento desse framework.



Exercícios de Fixação

- 1 Crie um projeto do tipo *Dynamic Web Project* chamado **K19-CRUD** seguindo os passos vistos nos exercícios 1 ao 5 do Capítulo 5.
- 2 Crie um pacote chamado `br.com.k19.model` e adicione nele uma classe para modelar os usuários da nossa aplicação.

```
1 package br.com.k19.model;
2
3 public class Usuario {
4     private Integer id;
5
6     private String nome;
7
8     private String username;
9
10    private String password;
11
12    // GETTERS E SETTERS
13 }
```

Código Java 9.1: `Usuario.java`

- 3 Implemente, no pacote `br.com.k19.model`, um repositório de usuários.

```
1 package br.com.k19.model;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class UsuarioRepository {
7     private static List<Usuario> usuarios = new ArrayList<Usuario>();
8
9     public Usuario busca(int id) {
10         return UsuarioRepository.usuarios.get(id - 1);
11     }
12
13     public void adiciona(Usuario usuario) {
14         usuario.setId(UsuarioRepository.usuarios.size() + 1);
15         UsuarioRepository.usuarios.add(usuario);
16     }
17
18     public void remove(int id) {
```

```
19     UsuarioRepository.usuarios.remove(id - 1);
20 }
21
22 public void altera(Usuario usuario) {
23     UsuarioRepository.usuarios.set(usuario.getId() - 1, usuario);
24 }
25
26 public List<Usuario> lista() {
27     return new ArrayList<Usuario>(UsuarioRepository.usuarios);
28 }
29 }
```

Código Java 9.2: *UsuarioRepository.java*

- 4 Crie um pacote chamado `br.com.k19.actions` e adicione nele uma classe para implementar as ações da nossa aplicação.

```
1 package br.com.k19.actions;
2
3 import java.util.List;
4 import br.com.k19.model.Usuario;
5 import br.com.k19.model.UsuarioRepository;
6 import com.opensymphony.xwork2.ActionSupport;
7
8 @SuppressWarnings("serial")
9 public class UsuarioAction extends ActionSupport {
10
11     private Usuario usuario = new Usuario();
12
13     private List<Usuario> usuarios;
14
15     public String adicionaOuAltera() {
16         UsuarioRepository repository = new UsuarioRepository();
17         if (this.usuario.getId() == null) {
18             repository.adiciona(this.usuario);
19         } else {
20             repository.altera(this.usuario);
21         }
22         this.usuario = new Usuario();
23         return UsuarioAction.SUCCESS;
24     }
25
26     public String remove() {
27         UsuarioRepository repository = new UsuarioRepository();
28         repository.remove(this.usuario.getId());
29         this.usuario = new Usuario();
30         return UsuarioAction.SUCCESS;
31     }
32
33     public String preparaAlteracao() {
34         UsuarioRepository repository = new UsuarioRepository();
35         this.usuario = repository.busca(this.usuario.getId());
36         return UsuarioAction.SUCCESS;
37     }
38
39     public String lista() {
40         UsuarioRepository repository = new UsuarioRepository();
41         this.usuarios = repository.lista();
42         return UsuarioAction.SUCCESS;
43     }
44
45     public Usuario getUsuario() {
46         return usuario;
47     }
48
49     public List<Usuario> getUsuarios() {
50         return usuarios;
```

```
51 }
52 }
```

Código Java 9.3: UsuarioAction.java

- 5** Registre as ações implementadas na classe UsuarioAction alterando o arquivo struts.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9     <action name="UsuarioAdicionaOuAltera" class="br.com.k19.actions.UsuarioAction"
10    method="adicionaOuAltera">
11      <result name="success" type="redirectAction">UsuarioLista</result>
12      <result name="input"/>/UsuarioFormulario.jsp</result>
13    </action>
14
15    <action name="UsuarioRemove" class="br.com.k19.actions.UsuarioAction"
16      method="remove">
17      <result name="success" type="redirectAction">UsuarioLista</result>
18    </action>
19
20    <action name="UsuarioPreparaAlteracao" class="br.com.k19.actions.UsuarioAction"
21      method="preparaAlteracao">
22      <result name="success"/>/UsuarioFormulario.jsp</result>
23    </action>
24
25    <action name="UsuarioLista" class="br.com.k19.actions.UsuarioAction"
26      method="lista">
27      <result name="success"/>/Lista.jsp</result>
28    </action>
29  </package>
30 </struts>
```

Código XML 9.1: struts.xml

- 6** Implemente a página inicial da nossa aplicação, adicionando o arquivo especificado abaixo na pasta WebContent.

```
1 <%@ taglib prefix="s" uri="/struts-tags"%>
2
3 <html>
4   <head>
5     <title>K19 Treinamentos</title>
6   </head>
7   <s:a action="UsuarioLista">Usuários</s:a>
8 </html>
```

Código JSP 9.1: Index.jsp

- 7** Implemente a página de listagem de usuários.

```
1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@ taglib prefix="s" uri="/struts-tags"%>
3
4 <html>
5   <head>
```

```

6   <title>K19 Treinamentos</title>
7 </head>
8
9 <s:a href="Index.jsp">Home</s:a>
10 <br/>
11 <s:a href="UsuarioFormulario.jsp">Novo Usuário</s:a>
12 <hr/>
13
14 <c:if test="${not empty usuarios}">
15   <h1>Usuários</h1>
16   <table>
17     <tr>
18       <th>ID</th>
19       <th>Nome</th>
20       <th>Username</th>
21       <th>Password</th>
22       <th>Alterar</th>
23       <th>Remover</th>
24     </tr>
25
26   <s:iterator value="usuarios" status="status">
27     <tr style="background-color: ${status.even ? '#EEEEEE' : '#FFFFFF'}">
28       <td><s:property value="id"/></td>
29       <td><s:property value="nome"/></td>
30       <td><s:property value="username"/></td>
31       <td><s:property value="password"/></td>
32       <td>
33         <s:a action="UsuarioPreparaAlteracao" >
34           alterar
35           <s:param name="usuario.id" value="id"/>
36         </s:a>
37       </td>
38       <td>
39         <s:a action="UsuarioRemove" >
40           remover
41           <s:param name="usuario.id" value="id"/>
42         </s:a>
43       </td>
44     </tr>
45   </s:iterator>
46 </table>
47 </c:if>
48
49 <c:if test="${empty usuarios}">
50   <h1>Sem usuários cadastrados.</h1>
51 </c:if>
52 </html>

```

Código JSP 9.2: Lista.jsp

Nas linhas em destaque, note que utilizamos a biblioteca de tags JSTL (JavaServer Pages Standard Tag Library). As tags dessa biblioteca não dependem de nenhum framework web em particular. Para saber mais sobre essa biblioteca, acesse <http://java.sun.com/jsp/jstl/>. Na página definida pelo arquivo acima, a tabela contendo os dados dos usuários só é exibida se a lista de usuários não for vazia. Para verificar se a lista de usuários não está vazia, utilizamos a tag `<c:if>` da JSTL. Alternativamente, poderíamos utilizar a tag `<s:if>` do Struts.

8 Implemente o formulário de cadastro ou alteração de usuários.

```

1 <%@ taglib prefix="s" uri="/struts-tags"%>
2
3 <html>
4   <head>
5     <title>K19 Treinamentos</title>
6     <s:head/>

```

```

7 </head>
8
9 <s:a href="Index.jsp">Home</s:a>
10 <hr/>
11
12 <h1>Cadastro ou Alteração de Usuários</h1>
13
14 <s:form action="UsuarioAdicionaOuAltera">
15   <s:hidden name="usuario.id"/>
16   <s:textfield label="Nome: " name="usuario.nome" />
17   <s:textfield label="Username: " name="usuario.username" />
18   <s:textfield label="Password: " name="usuario.password" />
19   <s:submit value="Salvar" />
20 </s:form>
21 </html>

```

Código JSP 9.3: UsuarioFormulario.jsp

- 9 Acesse a aplicação no endereço:

<http://localhost:8080/K19-CRUD/Index.jsp>

Teste as funcionalidades implementadas.

- 10 Adicione algumas regras de validação associadas ao cadastro e alteração de usuários. Crie um arquivo chamado **UsuarioAction-UsuarioAdicionaOuAltera-validation.xml** no pacote **br.com.k19.actions**. O nome desse arquivo é definido da seguinte forma:

[NomeDaClasse]-[NomeDaAction]-validation.xml

[NomeDaClasse] é o nome da classe que define as actions.

[NomeDaAction] é o nome da action registrado no arquivo struts.xml.

Acrescente o seguinte conteúdo a esse arquivo:

```

1 <!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
2   "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
3 <validators>
4   <field name="usuario.nome">
5     <field-validator type="requiredstring">
6       <message>O nome do usuário é obrigatório</message>
7     </field-validator>
8     <field-validator type="stringlength">
9       <param name="maxLength">30</param>
10      <message>O nome não pode possuir mais do que ${maxLength} letras.</message>
11    </field-validator>
12  </field>
13
14  <field name="usuario.username">
15    <field-validator type="requiredstring">
16      <message>O username do usuário é obrigatório</message>
17    </field-validator>
18    <field-validator type="stringlength">
19      <param name="maxLength">10</param>
20      <param name="minLength">5</param>
21      <message>O usuário deve possuir no mínimo ${minLength} e no máximo ${maxLength}<br>
22        letras.</message>
23    </field-validator>

```

```
23 </field>
24
25 <field name="usuario.password">
26   <field-validator type="requiredstring">
27     <message>A senha do usuário é obrigatória</message>
28   </field-validator>
29 </field>
30</validators>
```

Código XML 9.2: *UsuarioAction-UsuarioAdicionaOuAltera-validation.xml*

Observe que as validações são definidas individualmente para cada campo por meio da tag `<field>`. Associamos aos campos `usuario.nome`, `usuario.username` e `usuario.password` o validador `requiredstring`, tornando o preenchimento desses campos obrigatório. Também foram definidas mensagens de erro para essas validações através da tag `<message>`. O validador `stringlength` foi aplicado aos campos `usuario.nome` e `usuario.username`. Esse validador permite limitar o tamanho do texto desses campos.

- 11 Acesse a aplicação novamente no endereço:

<http://localhost:8080/K19-CRUD/Index.jsp>

Teste as validações no cadastro e na alteração de usuários.



PROJETO

Neste capítulo, criaremos uma aplicação que agrupa todos os recursos vistos nos capítulos anteriores.



Exercícios de Fixação

- 1 Crie um projeto do tipo *Dynamic Web Project* chamado **K19-Projeto** seguindo os passos vistos nos exercícios 1 ao 5 do Capítulo 5.
- 2 Adicione uma pasta chamada META-INF na pasta src do projeto K19-Projeto.
- 3 Configure o JPA adicionando o arquivo persistence.xml na pasta src/META-INF.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0"
3   xmlns="http://java.sun.com/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/←
6     ns/persistence/persistence_2_0.xsd">
7
8   <persistence-unit name="K19-PU" transaction-type="RESOURCE_LOCAL">
9     <provider>org.hibernate.ejb.HibernatePersistence</provider>
10    <properties>
11      <property name="hibernate.dialect" value="org.hibernate.dialect.←
12        MySQL5InnoDBDialect"/>
13
14      <property name="hibernate.hbm2ddl.auto" value="update"/>
15
16      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
17
18      <property name="javax.persistence.jdbc.user" value="root"/>
19
20      <property name="javax.persistence.jdbc.password" value="root"/>
21
22      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/←
23        K19_DB"/>
24    </properties>
25  </persistence-unit>
26 </persistence>
```

Código XML A.1: persistence.xml

- 4 Abra um terminal; entre no cliente do MySQL Server; apague se existir a base de dados K19_DB; e crie uma base de dados nova chamada K19_DB.

```
k19@k19-11:~/rafael$ mysql -u root -p
```

```

Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 36
Server version: 5.1.58-1ubuntu1 (Ubuntu)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> DROP DATABASE IF EXISTS K19_DB;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE DATABASE K19_DB;
Query OK, 1 row affected (0.02 sec)

```

- 5 Crie um pacote chamado br.com.k19.filters na pasta src do projeto K19-Projeto.
- 6 No pacote br.com.k19.filters, crie uma classe chamada JPAFilter com o seguinte conteúdo:

```

1 package br.com.k19.filters;
2
3 import java.io.IOException;
4 import javax.persistence.*;
5 import javax.servlet.*;
6
7 public class JPAFilter implements Filter {
8
9     private EntityManagerFactory factory;
10
11    @Override
12    public void init(FilterConfig filterConfig) throws ServletException {
13        this.factory = Persistence.createEntityManagerFactory("K19-PU");
14    }
15
16    @Override
17    public void destroy() {
18        this.factory.close();
19    }
20
21    @Override
22    public void doFilter(ServletRequest request, ServletResponse response,
23                         FilterChain chain) throws IOException, ServletException {
24
25        // CHEGADA
26        EntityManager manager = this.factory.createEntityManager();
27        request.setAttribute("EntityManager", manager);
28        manager.getTransaction().begin();
29        // CHEGADA
30
31        // FILTRO DO STRUTS
32        chain.doFilter(request, response);
33        // FILTRO DO STRUTS
34
35        // SAÍDA
36        try {
37            manager.getTransaction().commit();
38        } catch (Exception e) {
39            manager.getTransaction().rollback();
40        } finally {
41            manager.close();
42        }
43        // SAÍDA
44    }
45}

```

Código Java A.1: JPAFilter.java

- 7 Registre o JPAFilter alterando o arquivo web.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/←
4     javaee/web-app_2_5.xsd"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/←
6     javaee/web-app_3_0.xsd"
7   id="WebApp_ID" version="3.0">
8
9   <display-name>K19-Projeto</display-name>
10  <welcome-file-list>
11    <welcome-file>index.html</welcome-file>
12    <welcome-file>index.htm</welcome-file>
13    <welcome-file>index.jsp</welcome-file>
14    <welcome-file>default.html</welcome-file>
15    <welcome-file>default.htm</welcome-file>
16    <welcome-file>default.jsp</welcome-file>
17  </welcome-file-list>
18
19  <filter>
20    <filter-name>JPAFilter</filter-name>
21    <filter-class>br.com.k19.filters.JPAFilter</filter-class>
22  </filter>
23
24  <filter-mapping>
25    <filter-name>JPAFilter</filter-name>
26    <url-pattern>/*</url-pattern>
27  </filter-mapping>
28
29  <filter>
30    <filter-name>struts2</filter-name>
31    <filter-class>
32      org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
33    </filter-class>
34  </filter>
35
36  <filter-mapping>
37    <filter-name>struts2</filter-name>
38    <url-pattern>/*</url-pattern>
39  </filter-mapping>
40
41 </web-app>

```

Código XML A.2: web.xml

- 8 Crie uma classe chamada ControleDeAcesso no pacote br.com.k19.filters no projeto K19-Projeto com o seguinte conteúdo:

```

1 package br.com.k19.filters;
2
3 import java.io.IOException;
4
5 import javax.servlet.Filter;
6 import javax.servlet.FilterChain;
7 import javax.servlet.FilterConfig;
8 import javax.servlet.ServletException;
9 import javax.servlet.ServletRequest;
10 import javax.servlet.ServletResponse;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13 import javax.servlet.http.HttpSession;
14
15 public class ControleDeAcesso implements Filter {
16
17   @Override
18   public void doFilter(ServletRequest request, ServletResponse response,

```

```

19     FilterChain chain) throws IOException, ServletException {
20
21     HttpServletRequest req = (HttpServletRequest) request;
22     HttpSession session = req.getSession();
23
24     if (session.getAttribute("usuario") != null
25         || req.getRequestURI().endsWith("Login.jsp")
26         || req.getRequestURI().endsWith("Login.action")) {
27         chain.doFilter(request, response);
28     } else {
29         HttpServletResponse res = (HttpServletResponse) response;
30         res.sendRedirect("Login.jsp");
31     }
32 }
33
34 @Override
35 public void init(FilterConfig filterConfig) throws ServletException {
36
37 }
38
39 @Override
40 public void destroy() {
41
42 }
43 }
```

Código Java A.2: ControleDeAcesso.java

9 Registre o filtro ControleDeAcesso alterando o arquivo web.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee"
4   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/←
6       javaee/web-app_3_0.xsd"
7   id="WebApp_ID" version="3.0">
8
9   <display-name>K19-Projeto</display-name>
10  <welcome-file-list>
11    <welcome-file>index.html</welcome-file>
12    <welcome-file>index.htm</welcome-file>
13    <welcome-file>index.jsp</welcome-file>
14    <welcome-file>default.html</welcome-file>
15    <welcome-file>default.htm</welcome-file>
16    <welcome-file>default.jsp</welcome-file>
17  </welcome-file-list>
18
19  <filter>
20    <filter-name>Controle De Acesso</filter-name>
21    <filter-class>br.com.k19.filters.ControleDeAcesso</filter-class>
22  </filter>
23
24  <filter-mapping>
25    <filter-name>Controle De Acesso</filter-name>
26    <url-pattern>/*</url-pattern>
27  </filter-mapping>
28
29  <filter>
30    <filter-name>JPAPFilter</filter-name>
31    <filter-class>br.com.k19.filters.JPAPFilter</filter-class>
32  </filter>
33
34  <filter-mapping>
35    <filter-name>JPAPFilter</filter-name>
36    <url-pattern>/*</url-pattern>
37  </filter-mapping>
```

```

37 <filter>
38   <filter-name>struts2</filter-name>
39   <filter-class>
40     org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
41   </filter-class>
42 </filter>
43
44 <filter-mapping>
45   <filter-name>struts2</filter-name>
46   <url-pattern>/*</url-pattern>
47 </filter-mapping>
48
49 </web-app>

```

Código XML A.3: web.xml

- 10** Crie um pacote chamado br.com.k19.model e adicione nele uma classe para modelar os usuários da nossa aplicação.

```

1 package br.com.k19.model;
2
3 @Entity
4 public class Usuario {
5   @Id @GeneratedValue
6   private Integer id;
7
8   private String nome;
9
10  @Column(unique=true)
11  private String username;
12
13  private String password;
14
15  // GETTERS E SETTERS
16 }

```

Código Java A.3: Usuario.java

- 11** Implemente, no pacote br.com.k19.model, um repositório de usuários.

```

1 package br.com.k19.model;
2
3 public class UsuarioRepository {
4   private EntityManager manager;
5
6   public UsuarioRepository(EntityManager manager) {
7     this.manager = manager;
8   }
9
10  public Usuario busca(Integer id) {
11    return this.manager.find(Usuario.class, id);
12  }
13
14  public void adiciona(Usuario usuario) {
15    this.manager.persist(usuario);
16  }
17
18  public void remove(Integer id) {
19    Usuario usuario = this.manager.find(Usuario.class, id);
20    this.manager.remove(usuario);
21  }
22
23  public Usuario altera(Usuario usuario) {
24    return this.manager.merge(usuario);
25  }

```

```

25 }
26
27 public List<Usuario> lista() {
28     Query query = this.manager.createQuery("select x from Usuario x");
29     return query.getResultList();
30 }
31
32 public boolean existeUsernameEPassword(String username, String password) {
33     Query query = this.manager.createQuery(
34         "select x from Usuario x where x.username = :username and x.password = :←
35             password");
36     query.setParameter("username", username);
37     query.setParameter("password", password);
38     return !query.getResultList().isEmpty();
39 }
40 }
```

Código Java A.4: UsuarioRepository.java

- 12** Crie uma classe chamada AutenticacaoAction em um pacote chamado br.com.k19.actions no projeto K19-Projeto com o seguinte conteúdo:

```

1 package br.com.k19.actions;
2
3 // IMPORTS
4
5 @SuppressWarnings("serial")
6 public class AutenticacaoAction extends ActionSupport {
7
8     private Usuario usuario = new Usuario();
9
10    private String mensagem;
11
12    public String login() {
13        EntityManager manager = this.getEntityManager();
14        UsuarioRepository repository = new UsuarioRepository(manager);
15
16        if (repository.existeUsernameEPassword(
17            this.usuario.getUsername(), this.usuario.getPassword())) {
18
19            HttpSession session =
20                ServletActionContext.getRequest().getSession(true);
21            session.setAttribute("usuario", this.usuario);
22
23            this.mensagem = "Autenticação realizada com sucesso.";
24            return AutenticacaoAction.SUCCESS;
25        } else {
26            this.mensagem = "Usuário e/ou senha incorretos.";
27            return AutenticacaoAction.INPUT;
28        }
29    }
30
31    public String logout() {
32        HttpSession session = ServletActionContext.getRequest()
33            .getSession();
34        session.removeAttribute("usuario");
35
36        this.mensagem = "Até logo!";
37
38        return AutenticacaoAction.SUCCESS;
39    }
40
41    private EntityManager getEntityManager() {
42        HttpServletRequest request = ServletActionContext.getRequest();
43        EntityManager manager = (EntityManager)request.getAttribute("EntityManager");
44 }
```

```

45     return manager;
46 }
47
48 // GETTERS E SETTERS
49 }
```

Código Java A.5: AutenticacaoAction.java

- 13** Registre as actions criadas anteriormente alterando o arquivo struts.xml.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9     <action name="Login" class="br.com.k19.actions.AutenticacaoAction" method="login">
10    <result name="success">/Home.jsp</result>
11    <result name="input">/Login.jsp</result>
12  </action>
13  <action name="Logout" class="br.com.k19.actions.AutenticacaoAction" method="←
14    logout">
15    <result name="success">/Login.jsp</result>
16  </action>
17 </package>
</struts>
```

Código XML A.4: struts.xml

- 14** Para testar, crie a tela do formulário de autenticação e a tela principal da aplicação na pasta WebContent.

```

1 <%@ taglib prefix="s" uri="/struts-tags"%>
2 <html>
3   <head>
4     <title>K19 Treinamentos</title>
5   </head>
6   <body>
7     <div style="color: red">
8       <s:property value="mensagem"/>
9     </div>
10
11    <s:form action="Login">
12      <s:textfield name="usuario.username" label="Usuário: " />
13      <s:password name="usuario.password" label="Senha: " />
14      <s:submit value="Enviar" />
15    </s:form>
16  </body>
17 </html>
```

Código JSP A.1: Login.jsp

```

1 <%@ taglib prefix="s" uri="/struts-tags"%>
2 <html>
3   <head>
4     <title>K19 Treinamentos</title>
5   </head>
6   <body>
7     <div style="color: red">
8       <s:property value="mensagem"/>
9     </div>
```

```
10 <s:a href="Home.jsp">Home</s:a>
11 <br/>
12 <s:a action="Logout">Logout</s:a>
13 </body>
14 </html>
```

Código JSP A.2: Home.jsp

- 15 Adicione a aplicação no Glassfish e inicialize-o. Acesse o terminal do MySQL e cadastre o primeiro usuário da aplicação. Siga os passos abaixo.

```
k19@k19-11:~/rafael$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 5.1.58-1ubuntu1 (Ubuntu)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE K19_DB;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> INSERT INTO Usuario(nome,username,password) VALUES('admin','admin','admin');
Query OK, 1 row affected (0.03 sec)
```

- 16 Teste a aplicação acessando a página:

<http://localhost:8080/K19-Projeto/Home.jsp>

- 17 No pacote br.com.k19.actions, adicione uma classe para implementar o cadastro, edição, remoção e visualização dos usuários.

```
1 package br.com.k19.actions;
2
3 import java.util.List;
4
5 import br.com.k19.model.Usuario;
6 import br.com.k19.model.UsuarioRepository;
7
8 import com.opensymphony.xwork2.ActionSupport;
9
10 @SuppressWarnings("serial")
11 public class UsuarioAction extends ActionSupport {
12
13     private Usuario usuario = new Usuario();
14
15     private List<Usuario> usuarios;
16
17     public String adicionaOuAltera() {
18         EntityManager manager = this.getEntityManager();
19         UsuarioRepository repository = new UsuarioRepository(manager);
20         if (this.usuario.getId() == null) {
21             repository.adiciona(this.usuario);
22         } else {
23             repository.altera(this.usuario);
24         }
25         this.usuario = new Usuario();
```

```

26     return UsuarioAction.SUCCESS;
27 }
28
29 public String remove() {
30     EntityManager manager = this.getEntityManager();
31     UsuarioRepository repository = new UsuarioRepository(manager);
32     repository.remove(this.usuario.getId());
33     this.usuario = new Usuario();
34     return UsuarioAction.SUCCESS;
35 }
36
37 public String preparaAlteracao() {
38     EntityManager manager = this.getEntityManager();
39     UsuarioRepository repository = new UsuarioRepository(manager);
40     this.usuario = repository.busca(this.usuario.getId());
41     return UsuarioAction.SUCCESS;
42 }
43
44 public String lista() {
45     EntityManager manager = this.getEntityManager();
46     UsuarioRepository repository = new UsuarioRepository(manager);
47     this.usuarios = repository.lista();
48     return UsuarioAction.SUCCESS;
49 }
50
51 private EntityManager getEntityManager() {
52     HttpServletRequest request = ServletActionContext.getRequest();
53     EntityManager manager = (EntityManager)request.getAttribute("EntityManager");
54
55     return manager;
56 }
57
58 public Usuario getUsuario() {
59     return usuario;
60 }
61
62 public List<Usuario> getUsuarios() {
63     return usuarios;
64 }
65 }
```

Código Java A.6: UsuarioAction.java

18 Registre as ações implementadas na classe UsuarioAction alterando o arquivo struts.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
8   <package name="default" extends="struts-default">
9     <action name="Login" class="br.com.k19.actions.AutenticacaoAction" method="login">
10       <result name="success"/>/Home.jsp</result>
11       <result name="input"/>/Login.jsp</result>
12     </action>
13     <action name="Logout" class="br.com.k19.actions.AutenticacaoAction" method="logout">
14       <result name="success"/>/Login.jsp</result>
15     </action>
16
17     <action name="UsuarioAdicionaOuAltera" class="br.com.k19.actions.UsuarioAction" method="adicionaOuAltera">
18       <result name="success" type="redirectAction">UsuarioLista</result>
19       <result name="input"/>/UsuarioFormulario.jsp</result>
```

```

20   </action>
21
22   <action name="UsuarioRemove" class="br.com.k19.actions.UsuarioAction" method="→
23     remove">
24     <result name="success" type="redirectAction">UsuarioLista</result>
25   </action>
26
27   <action name="UsuarioPreparaAlteracao" class="br.com.k19.actions.UsuarioAction" →
28     method="preparaAlteracao">
29     <result name="success">/UsuarioFormulario.jsp</result>
30   </action>
31
32   <action name="UsuarioLista" class="br.com.k19.actions.UsuarioAction" method="→
33     lista">
34     <result name="success">/Lista.jsp</result>
35   </action>
36
37 </package>
38 </struts>

```

Código XML A.5: struts.xml

- 19** Altere a página Home.jsp adicionando um link para a listagem de usuários.

```

1 <%@ taglib prefix="s" uri="/struts-tags"%>
2 <html>
3   <head>
4     <title>K19 Treinamentos</title>
5   </head>
6   <body>
7     <div style="color: red">
8       <s:property value="mensagem"/>
9     </div>
10
11    <s:a href="Home.jsp">Home</s:a>
12    <br/>
13    <s:a action="UsuarioLista">Usuários</s:a>
14    <br/>
15    <s:a action="Logout">Logout</s:a>
16  </body>
17 </html>

```

Código JSP A.3: Home.jsp

- 20** Implemente a página de listagem de usuários.

```

1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@ taglib prefix="s" uri="/struts-tags"%>
3
4 <html>
5   <head>
6     <title>K19 Treinamentos</title>
7   </head>
8
9   <s:a href="Home.jsp">Home</s:a>
10  <br/>
11  <s:a href="UsuarioFormulario.jsp">Novo Usuário</s:a>
12  <br/>
13  <s:a action="Logout">Logout</s:a>
14  <hr/>
15
16  <c:if test="${not empty usuarios}">
17    <h1>Usuários</h1>
18    <table>

```

```

19   <tr>
20     <th>ID</th>
21     <th>Nome</th>
22     <th>Username</th>
23     <th>Password</th>
24     <th>Alterar</th>
25     <th>Remover</th>
26   </tr>
27
28   <s:iterator value="usuarios" status="status">
29     <tr style="background-color: ${status.even ? '#EEEEEE' : '#FFFFFF'}">
30       <td><s:property value="id"/></td>
31       <td><s:property value="nome"/></td>
32       <td><s:property value="username"/></td>
33       <td><s:property value="password"/></td>
34       <td>
35         <s:a action="UsuarioPreparaAlteracao" >
36           alterar
37           <s:param name="usuario.id" value="id"/>
38         </s:a>
39       </td>
40       <s:a action="UsuarioRemove" >
41         remover
42         <s:param name="usuario.id" value="id"/>
43       </s:a>
44     </td>
45   </tr>
46 </s:iterator>
47 </table>
48 </c:if>
49
50 <c:if test="${empty usuarios}">
51   <h1>Sem usuários cadastrados.</h1>
52 </c:if>
53
54 </html>

```

Código JSP A.4: Lista.jsp

21 Implemente o formulário de cadastro ou alteração de usuários.

```

1 <%@ taglib prefix="s" uri="/struts-tags"%>
2
3 <html>
4   <head>
5     <title>K19 Treinamentos</title>
6     <s:head/>
7   </head>
8
9   <s:a href="Home.jsp">Home</s:a>
10  <br/>
11  <s:a action="Logout">Logout</s:a>
12  <hr/>
13
14  <h1>Cadastro ou Alteração de Usuários</h1>
15
16  <s:form action="UsuarioAdicionaOuAltera">
17    <s:hidden name="usuario.id"/>
18    <s:textfield label="Nome" name="usuario.nome" />
19    <s:textfield label="Username" name="usuario.username"/>
20    <s:textfield label="Password" name="usuario.password"/>
21    <s:submit value="Salvar"/>
22  </s:form>
23 </html>

```

Código JSP A.5: UsuarioFormulario.jsp

- 22 Acesse a aplicação no endereço:

<http://localhost:8080/K19-Projeto/Home.jsp>

Cadastre, altere, remova alguns usuários.

- 23 Adicione algumas regras de validação associadas ao cadastro e alteração de usuários. Crie um arquivo chamado **UsuarioAction-UsuarioAdicionaOuAltera-validation.xml** no pacote **br.com.k19.actions** com o seguinte conteúdo.

```
1 <!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
2   "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
3 <validators>
4   <field name="usuario.nome">
5     <field-validator type="requiredstring">
6       <message>O nome do usuário é obrigatório</message>
7     </field-validator>
8     <field-validator type="stringlength">
9       <param name="maxLength">30</param>
10      <message>O nome não pode possuir mais do que ${maxLength} letras.</message>
11    </field-validator>
12  </field>
13  <field name="usuario.username">
14    <field-validator type="requiredstring">
15      <message>O username do usuário é obrigatório</message>
16    </field-validator>
17    <field-validator type="stringlength">
18      <param name="maxLength">10</param>
19      <param name="minLength">5</param>
20      <message>O usuário deve possuir no mínimo ${minLength} e no máximo ${maxLength}←
21        letras.</message>
22    </field-validator>
23  </field>
24  <field name="usuario.password">
25    <field-validator type="requiredstring">
26      <message>A senha do usuário é obrigatória</message>
27    </field-validator>
28  </field>
29 </validators>
```

Código XML A.6: *UsuarioAction-UsuarioAdicionaOuAltera-validation.xml*

- 24 Acesse a aplicação novamente no endereço:

<http://localhost:8080/K19-Projeto/Home.jsp>

Teste as validações no cadastro e na alteração de usuários.

- 25 Analogamente ao que foi feito até agora, crie telas para cadastrar, editar, remover e consultar deputados estaduais. Para cada deputado, o sistema deve armazenar o seu nome, partido, telefone e e-mail.