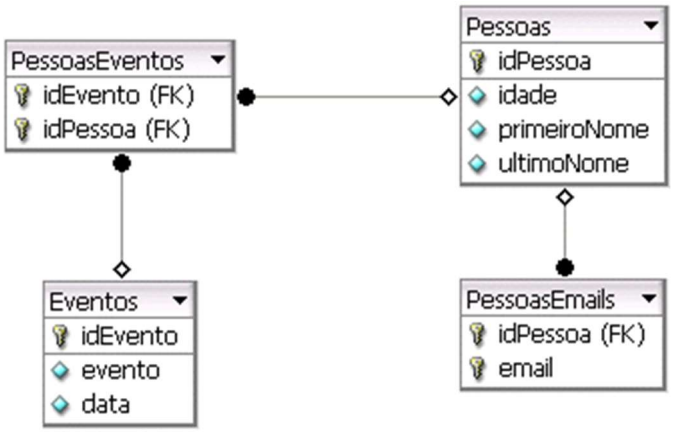
Associações, Coleções e Herança

Modelo de Dados

Para esta aula iremos nos basear no seguinte modelo de dados



Mapeando associações

Na aula anterior nós mapeamos uma classe de entidade de persistência para uma tabela.

Agora vamos continuar e adicionar algumas associações de classe.

Primeiro nós iremos adicionar pessoas a nossa aplicação, e armazenar os eventos de que elas participam

Mapeando a classe Eventos

Crie e compile a classe Eventos

Eventos.java

```java
package br.ifmt.dai.curso;
public class Eventos {
    public Eventos(){
    }
    private int idEvento=0;
    private String evento="";
    private String dataEvento="";
    public int getIdEvento() {
        return idEvento;
    }
    public void setIdEvento(int idEvento) {
        this.idEvento = idEvento;
    }
    public String getEvento() {
        return evento;
    }
    public void setEvento(String evento) {
        this.evento = evento;
    }
    public String getDataEvento() {
        return dataEvento;
    }
    public void setDataEvento(String dataEvento) {
        this.dataEvento = dataEvento;
    }
}
```

Mapeando a classe Pessoas

Crie e compile a classe Pessoas

Pessoas.java

```java
package br.ifmt.dai.curso;
public class Pessoas {
    public Pessoas(){
    }
    private int idPessoa;
    private int idade;
    private String primeiroNome;
    private String ultimoNome;
    public int getIdPessoa() {
        return idPessoa;
    }
    public void setIdPessoa(int idPessoa) {
        this.idPessoa = idPessoa;
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    public String getPrimeiroNome() {
        return primeiroNome;
    }
    public void setPrimeiroNome(String primeiroNome)
        this.primeiroNome = primeiroNome;
    }
    public String getUltimoNome() {
        return ultimoNome;
    }
    public void setUltimoNome(String ultimoNome) {
        this.ultimoNome = ultimoNome;
    }
}
```

Arquivos de mapeamentos

Crie um novo arquivo de mapeamento, chamado **Eventos.hbm.xml**

Eventos.hbm.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="br.ifmt.dai.curso.Eventos" table="eventos" >
        <id name="idEvento" column="idEvento" type="int">
            <generator class="native"/>
        </id>
        <property name="evento" column="evento" type="string"/>
        <property name="dataEvento" column="data" type="string"/>
    </class>
</hibernate-mapping>
```

Mapeamento Pessoas

Crie um novo arquivo de mapeamento, chamado Pessoas.hbm.xml

Pessoas.hbm.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="br.ifmt.da.curso.Pessoas" table="pessoas" >
        <id name="idPessoa" column="idPessoa" type="int">
            <generator class="native"/>
        </id>
        <property name="idade" column="idade"/>
        <property name="primeiroNome" column="primeiroNome"/>
        <property name="ultimoNome" column="ultimoNome"/>
    </class>
</hibernate-mapping>
```

Configuração do Hibernate

Finalmente, adicione os novos mapeamentos a configuração do Hibernate

hibernate.cfg.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/escola</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">ifmt2k17</property>
    <property name="hibernate.connection.pool_size">1</property>
    <property name="hibernate.current_session_context_class">thread</property>
    <property name="hibernate.cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
    <property name="hibernate.show_sql">true</property>

    <mapping resource="br/ifmt/dai/curso/Eventos.hbm.xml"/>
    <mapping resource="br/ifmt/dai/curso/Pessoas.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Criando uma classe para gerenciar eventos

Crie e compile a classe GerenciadorEventos

GerenciadorEventos.java

```
package br.ifmt.dai.curso;
import ...3 linhas
public class GerenciadorEventos {
    public void salvarEvento(String evento, String data){
        try{
            SessionFactory sf=new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
            Session s= sf.openSession();
            Transaction tx=s.beginTransaction();
            Eventos e=new Eventos();
            e.setDataEvento(data);
            e.setEvento(evento);
            s.saveOrUpdate(e);
            tx.commit();
            s.close();
            System.out.println("Evento incluído");
        }catch(Exception x){
            System.out.println("Erro: "+x.getMessage());
        }
    }
    public List listarEventos(){
        SessionFactory sf=new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
        Session s= sf.openSession();
        s.beginTransaction();
        List lista=s.createQuery("from Eventos").list();
        s.getTransaction().commit();
        return lista;
    }
```

Criando a aplicação web

Criando o Servlet

GravarEventos.java

```
package br.ifmt.dai.curso;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class GravarEventos extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String evento =request.getParameter("evento");
        String dataEvento =request.getParameter("dataEvento");
        response.setContentType("text/html;charset=ISO-8859-1");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Gravando Eventos no Hibernate</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h2>Gravando Eventos no Hibernate</h2>");
            out.println("evento: "+evento+"<br>");
            out.println("Data: "+dataEvento+"<br>");
            GerenciadorEventos c=new GerenciadorEventos();
            c.salvarEvento(evento, dataEvento);
            out.println("</body>");
            out.println("</html>");
```

```
28          }
29      }
30      @Override
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
32              throws ServletException, IOException {
33          processRequest(request, response);
34      }
35  /*    @Override
36      protected void doPost(HttpServletRequest request, HttpServletResponse response)
37              throws ServletException, IOException {
38          processRequest(request, response);
39      } */
40  }
```

Página JSP

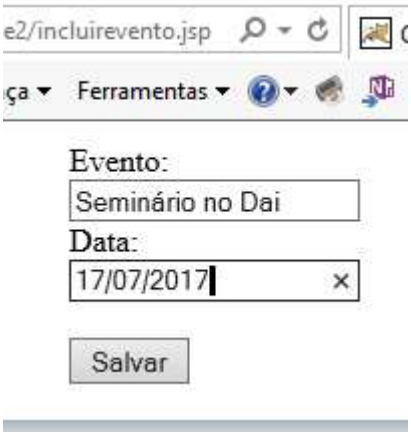
A página incluirevento.jsp vai chamar o Servlet

incluirevento.jsp

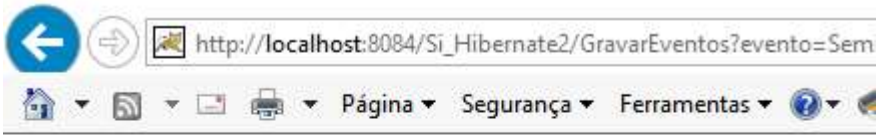
```
1  <%@page contentType="text/html"%>
2  <%@page pageEncoding="ISO-8859-1"%>
3      <head>
4          <title>Gravando Eventos no Hibernate</title>
5          <LINK href="estilo.css" type="text/css" rel="stylesheet">
6      </head>
7  <html>
8      <body>
9          <table border="0" align="center">
10             <tr><td>
11                 <form action="gravarEvento">
12                     Evento:<br><input type="text" name="evento" value="" /><br>
13                     Data:<br><input type="text" name="dataEvento" value="" /><br>
14                     <br><input type="submit" value="Salvar" name="incluir" />
15                 </form>
16             </td></tr>
17         </table>
18     </body>
19 </html>
```

Verificando a página para incluir um evento

Reinicie o Tomcat e execute a página incluirevento.jsp



Ao clicar no botão Salvar

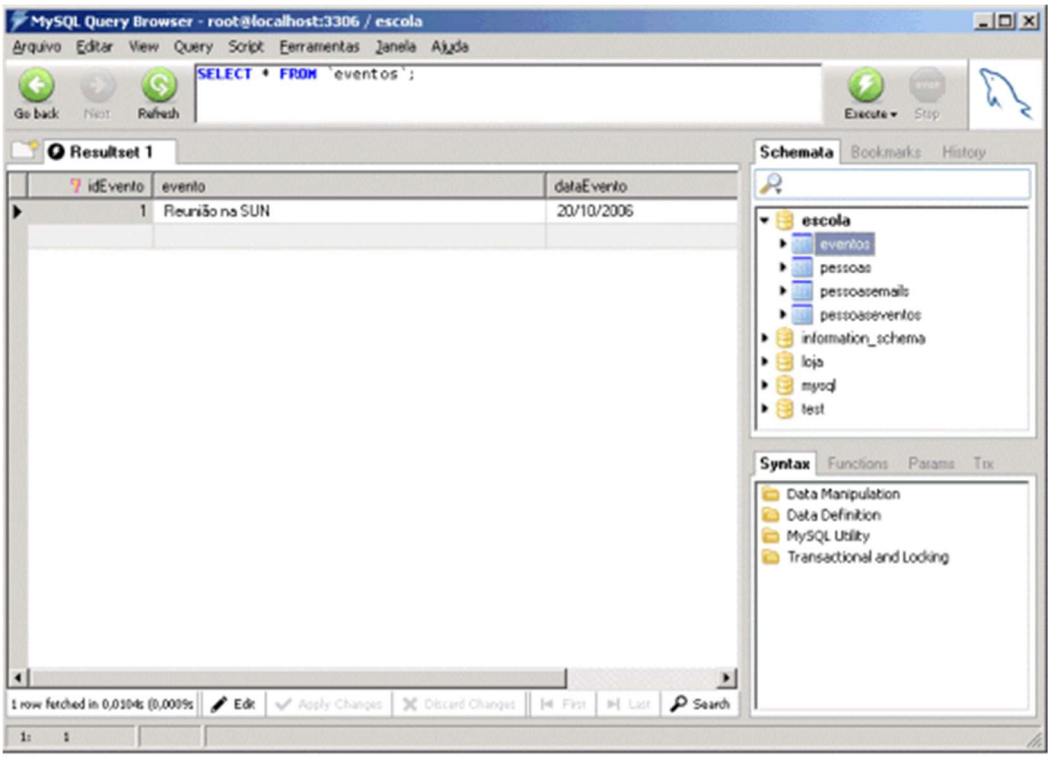


Gravando Eventos no Hibernate

evento: Semin?rio no Dai
Data: 17/07/2017

Verificação no Banco de Dados

Verifique se o evento foi gravado corretamente no MySQL



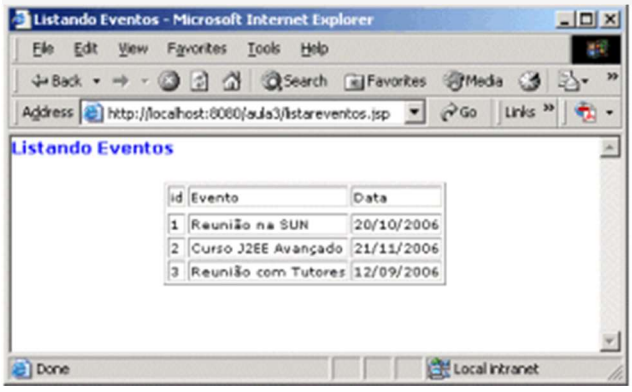
Listando os Eventos

Criando a página para listar eventos

listareventos.jsp

```jsp
<%@page contentType="text/html"%>
<%@page pageEncoding="ISO-8859-1"%>
<%@page import="org.hibernate.*"%>
<%@page import="org.hibernate.cfg.*"%>
<%@page import="br.com.wincomp.curso.*"%>
<%@page import="java.util.*"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <LINK href="estilo.css" type="text/css" rel="stylesheet">
        <title>Listando Eventos</title>
    </head>
    <body>
        <h2>Listando Eventos</h2>
        <%
            GerenciadorEventos ge = new GerenciadorEventos();
            List lista = ge.listarEventos();
            String tabela = "";
            tabela += "<table align=center border=1>";
            tabela += "<tr><td>id</td><td>Evento</td><td>Data</td><tr>";
            for(int i=0; i<lista.size();i++){
                Eventos c = (Eventos)lista.get(i);
                int idEvento = c.getIdEvento();
                String evento = c.getEvento();
                String data = c.getDataEvento();
                tabela += "<tr>";
                tabela += "<td>" + idEvento + "</td>";
                tabela += "<td>" + evento + "</td>";
                tabela += "<td>" + data + "</td>";
                tabela += "</tr>";
            }
            tabela += "</table>";
            out.println(tabela);
        %>
    </body>
</html>
```

Resultado



Observação

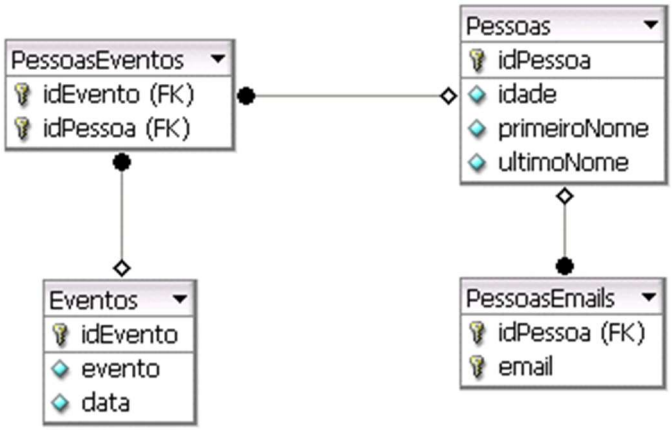
Neste exemplo utilizamos o HQL, a linguagem de Queries do Hibernate.

```
s.createQuery("from Eventos").list();
```

Veremos HQL, em mais detalhes mais a frente

Criando associações

Modelo de Dados



Nos agora criaremos uma associação entre estas duas entidades.

Obviamente, pessoas podem participar de eventos, e eventos possuem participantes.

As questões de design com que teremos de lidar são: direcionalidade, multiplicidade e comportamento de coleção.

Uma associação unidirectional baseada em um Set

Nos iremos adicionar uma coleção de eventos na classe Pessoa.

Desse jeito poderemos navegar pelos eventos de uma pessoa em particular, sem executar uma query explicitamente – apenas chamando **c.getEvents().**

Nos usaremos uma coleção Java, um **Set**, porquê a coleção não conterá elementos duplicados e a ordem não é relevante para nós.

Vamos adicionar o código abaixo à nossa classe Java para Pessoas e então fazer o mapeamento:

```
private Set events = new HashSet();

public Set getEvents() {
    return events;
}

public void setEvents(Set events) {
    this.events = events;
}
```

Uma associação unidirectional baseada em um Set

Nos iremos adicionar uma coleção de eventos na classe Pessoa.

Desse jeito poderemos navegar pelos eventos de uma pessoa em particular, sem executar uma query explicitamente – apenas chamando **c.getEvents().**

Nos usaremos uma coleção Java, um **Set**, porquê a coleção não conterá elementos duplicados e a ordem não é relevante para nós.

Vamos adicionar o código abaixo à nossa classe Java para Pessoas e então fazer o mapeamento:

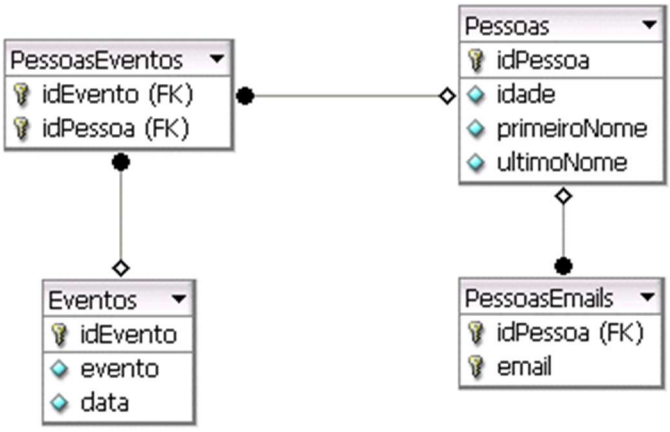
```
private Set events = new HashSet();

public Set getEvents() {
    return events;
}

public void setEvents(Set events) {
    this.events = events;
}
```

Mapeamento muitos-para-muitos

Veja novamente esquema de mapeamento para o banco de dados:



Daqui para frente, nós usaremos o mapeamento muitos-para-muitos do Hibernate:

Pessoas.hbm.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="br.ifmt.da.curso.Pessoas" table="pessoas" >
        <id name="idPessoa" column="idPessoa" type="int">
            <generator class="native"/>
        </id>
        <property name="idade" column="idade"/>
        <property name="primeiroNome" column="primeiroNome"/>
        <property name="ultimoNome" column="ultimoNome"/>
        <set name="events" table="pessoaseventos">
            <key column="idPessoa"/>
            <many-to-many column="idEvento" class="br.ifmt.dai.curso.Eventos"/>
        </set>
    </class>
</hibernate-mapping>

```

Observações sobre Mapeamento

O Hibernate suporta todo tipo de mapeamento de coleção, sendo um **<set>** mais comum.

Para uma associação muitos-para-muitos (ou relacionamento de entidade *n:m*), uma tabela associativa é necessária.

Em nosso modelo a tabela associativa é a tabela PessoasEventos

Cada linha nessa tabela representa um link entre uma pessoa e um evento.

O nome da tabela é configurado com o atributo *table* do elemento set.

```
<set name="events" table="PessoasEventos">
```

O nome da coluna identificadora na associção, pelo lado da pessoa, é definido com o elemento **<key>**

```
<key column="idPessoa"/>
```

O nome da coluna pelo lado dos eventos, e definido com o atributo **column** do **<many-to-many>**.

```
<many-to-many column="idEvento" class="br.ifmt.dai.curso.Eventos"/>
```

Você também precisa dizer para o Hibernate a classe dos objetos na sua coleção (a classe do outro lado das coleções de referência).

Trabalhando a associação

Vamos criar um novo método na classe GerenciadorEventos.java:

```
public void adicionarPessoasEventos(int idPessoa, int idEvento) {

    SessionFactory sf = new Configuration().configure("hibernate.cfg.xml").buildSessio
    Session s = sf.openSession();
    s.beginTransaction();
    Pessoas p = (Pessoas) s.load(Pessoas.class, idPessoa);
    Eventos e = (Eventos)s.load(Eventos.class, idEvento);
    p.getEvents().add(e);
    s.getTransaction().commit();
}
```

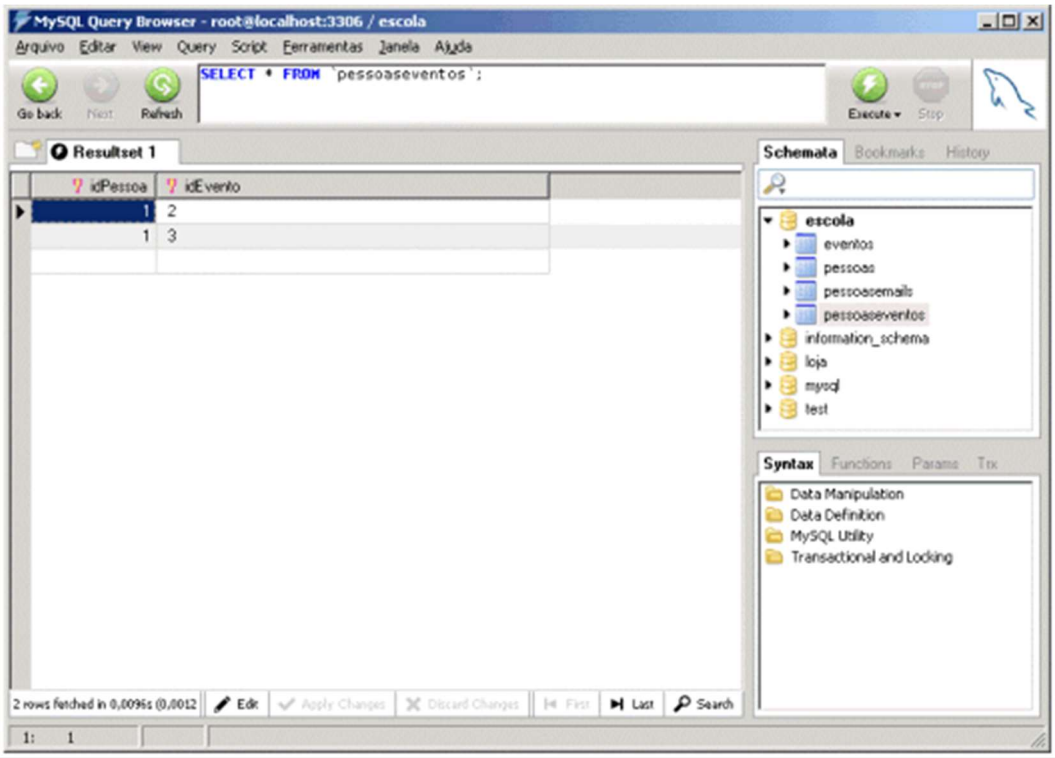
Para executar, à partir de uma página JSP, bastaria um código parecido com este abaixo

```
<%
    GerenciadorEventos ge = new GerenciadorEventos();
    ge.adicionarPessoasEventos(1,2);
%>
```

Como você pode ver, **não há chamada explícita para update() ou save()**, o Hibernate detecta automaticamente que a coleção foi modificada e precisa ser atualizada.

Isso é chamado de ***checagem automática de sujeira***, e você também pode usá-la modificando o nome ou a data de qualquer um dos seus objetos.

Verificando no Banco de Dados



Código completo da classe GerenciadorEventos

GerenciadorEventos.java

```
1  package br.ifmt.dai.curso;
2  import java.util.List;
3  import org.hibernate.*;
4  import org.hibernate.cfg.*;
5  public class GerenciadorEventos {
6      public void salvarEvento(String evento, String data){...17 linhas }
23     public List listarEventos(){
24         SessionFactory sf=new Configuration().configure("hibernate.cfg.xml").buildSessionFactory()
25         Session s= sf.openSession();
26         s.beginTransaction();
27         List lista=s.createQuery("from Eventos").list();
28         s.getTransaction().commit();
29         return lista;
30     }
31     public void adicionarPessoasEventos(int idPessoa,int idEvento){
32         SessionFactory sf=new Configuration().configure("hibernate.cfg.xml").buildSessionFactory()
33         Session s= sf.openSession();
34         s.beginTransaction();
35         Pessoas p=(Pessoas) s.load(Pessoas.class, idPessoa);
36         Eventos e=(Eventos)s.load(Eventos.class, idEvento);
37         p.getEvents().add(e);
38         s.getTransaction().commit();
39     }
40 }
41
```

Coleção de valores

Nós adicionamos uma coleção de objetos de tipo de valores à entidade Pessoas.

Nós querermos armazenar endereços de e-mail, para isso utilizamos o tipo String, e a coleção novamente será um **Set**:

Criando o método para adicionar email

Método adicionado à classe **GerenciadorEventos**

```
public void adicionarEmailPessoas(int idPessoa, String email) {

    SessionFactory sf = new  Configuration().configure("hibernate.cfg.xml").buildSessior
    Session s = sf.openSession();
    s.beginTransaction();
    Pessoas p = (Pessoas) s.load(Pessoas.class, idPessoa);
    p.getEmail().add(email);
    s.getTransaction().commit();
}
```

Código completo da classe

GerenciadorEventos.java

```
1   package br.ifmt.dai.curso;
2   import java.util.List;
3   import org.hibernate.*;
4   import org.hibernate.cfg.*;
5   public class GerenciadorEventos {
6       public void salvarEvento(String evento, String data){...17 linhas }
23      public List listarEventos(){...8 linhas }
31      public void adicionarPessoasEventos(int idPessoa,int idEvento){
32          SessionFactory sf=new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
33          Session s= sf.openSession();
34          s.beginTransaction();
35          Pessoas p=(Pessoas) s.load(Pessoas.class, idPessoa);
36          Eventos e=(Eventos)s.load(Eventos.class, idEvento);
37          p.getEvents().add(e);
38          s.getTransaction().commit();
39      }
40      public void adicionarEmailPessoas(int idPessoa, String email){
41          SessionFactory sf=new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
42          Session s= sf.openSession();
43          s.beginTransaction();
44          Pessoas p=(Pessoas) s.load(Pessoas.class, idPessoa);
45          p.getEmail().add(email);
46          s.getTransaction().commit();
47      }
48  }
```

Mapeamento da Coleção

O mapeamento deste **Set** no **Pessoas.hbm.xml** :

```
<set name="email" table="PessoasEmails">
    <key column="idPessoa"/>
    <element type="string" column="email"/>
</set>
```

Comparando com o mapeamento anterior a diferença se encontra na parte **element**, que indica ao Hibernate que a coleção não contém referências à outra entidade, mas uma coleção de elementos do tipo **String**

Mais uma vez, o atributo **table**do elemento **set** determina o nome da tabela para a coleção.

O elemento **key**define o nome da coluna foreign key na tabela de coleção.

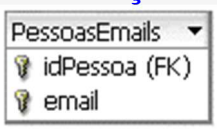
O atributo **column** dentro do elemento **element** define o nome da coluna onde os valores da String serão armazenados.

Código completo

Pessoas.hbm.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="br.ifmt.da.curso.Pessoas" table="pessoas" >
        <id name="idPessoa" column="idPessoa" type="int">
            <generator class="native"/>
        </id>
        <property name="idade" column="idade"/>
        <property name="primeiroNome" column="primeiroNome"/>
        <property name="ultimoNome" column="ultimoNome"/>
        <set name="events" table="pessoaseventos">
            <key column="idPessoa"/>
            <many-to-many column="idEvento" class="br.ifmt.dai.curso.Eventos"/>
        </set>
        <set name="email" table="pessoasemails">
            <key column="idPessoa"/>
            <element type="string" column="email"/>
        </set>
    </class>
</hibernate-mapping>
```

Observações



Você pode observar que a primary key da tabela da coleção é de na verdade uma chave composta, usando ambas as colunas.

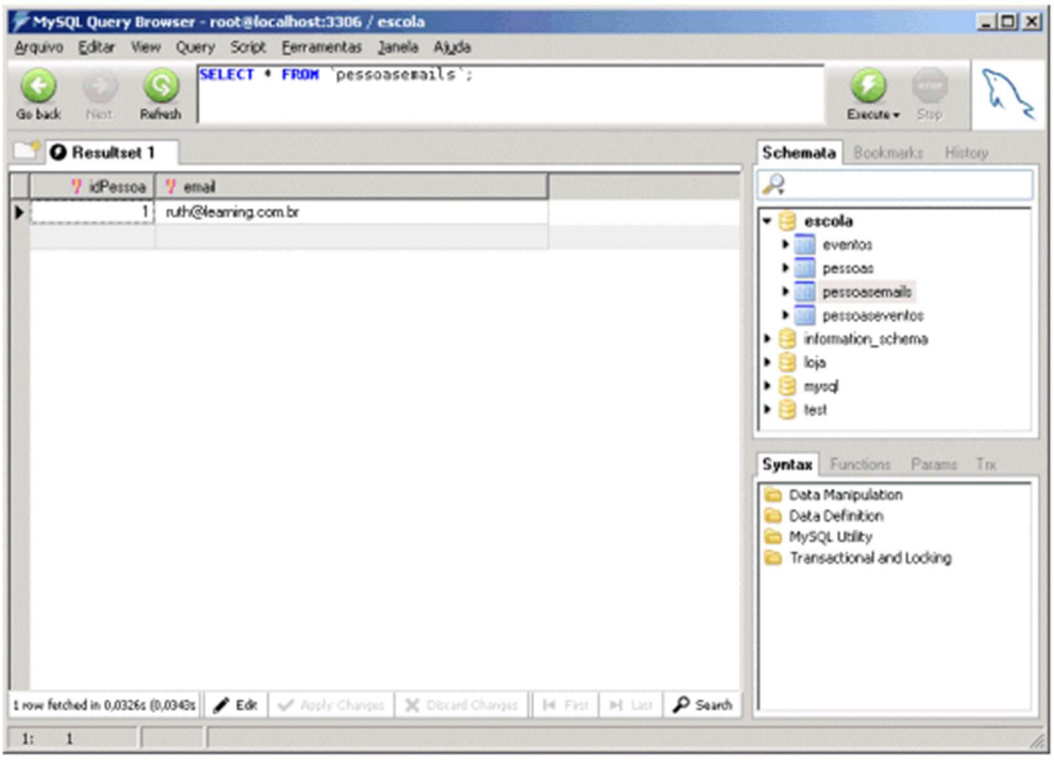
Isso também implica que cada pessoa não pode ter endereços de e-mail duplicados, o que é exatamente a semântica que precisamos para um set em Java.

Você pode agora tentar adicionar elementos a essa coleção, do mesmo modo que fizemos anteriormente ligando pessoas e eventos.

È o mesmo código em Java:

```
<%
    GerenciadorEventos ge = new GerenciadorEventos();
    ge.adicionarEmailPessoas(1, "ruth@learning.com.br");
%>
```

Verificando no Banco de Dados



Associações bidirecionais

Agora iremos mapear uma associação bidirecional,
fazendo a associação entre pessoas e eventos, de ambos os lados, em Java.

Logicamente, o esquema do banco de dados não muda,
nós continuamos tendo multiplicidades muitos-para-muitos.

Um banco de dados é mais flexível do que uma linguagem de programação, ele não precisa de nenhuma direção de navegação

Os dados podem ser acessados de qualquer forma.

Primeiramente, adicione uma coleção de participantes à classe Eventos

```
private Set participantes = new HashSet();

public Set getParticipantes() {
    return participantes;
}
public void setParticipantes(Set participantes) {
    this.participantes = participantes;
}
```

Agora mapeie este lado da associação em **Eventos.hbm.xml**

```
<set name="participantes" table="pessoaseventos" inverse="true" >
    <key column="idEvento"/>
    <many-to-many column="idPessoa" class="br.ifmt.da.curos.Pessoas"/>
</set>
```

Como você pode ver, esse é um mapeamento normal usando **set** em ambos documentos de mapeamento.

Observe que o nome das colunas em **key**e **many-to-many**estão trocados em ambos os documentos de mapeamento.

A adição mais importante feita está no atributo **inverse="true**" no elemento **set** do mapeamento da coleção da classe Eventos.

Isso significa que o Hibernate deve pegar o outro lado, a classe Pessoas quando necessitar encontrar informação sobre a relação entre as duas entidades.

Trabalhando com associações bidirecionais

Primeiro tenha em mente que o Hibernate não afeta a semântica normal do Java.

Como nós criamos uma associação entre uma **Pessoa**e um **Evento** no exemplo unidirecional?

Nós adicionamos uma instância de **Eventos**, da coleção de referências de eventos, a uma instância de **Pessoas.**

Então, obviamente, se nós quizermos que esta associação funcione bidirecionalmente, nós devemos fazer a mesma coisa do outro lado, adicionando uma referência de **Pessoas** na coleção de um **Evento**.

Esse acerto de associações de ambos os lados é absolutamente necessário e você nunca deve esquecer de faze-lo.

Muitos desenvolvedores programam de maneira defensiva e criam métodos gerenciadores de associações que ajustam corretamente ambos os lados:

```
protected Set events = new HashSet();

protected Set getEvents() {
    return events;
}
protected void setEvents(Set events) {
    this.events = events;
}

public void adicionarParaEventos(Eventos event) {
    this.getEvents().add(event);
    event.getParticipantes().add(this);
}
public void removerDeEventos(Eventos event) {
    this.getEvents().remove(event);
    event.getParticipantes().remove(this);
}
```

Observe que os métodos set e get da a coleção estão protegidos

Isso permite que classes e subclasses do mesmo pacote continuem acessando os métodos, mas previne que qualquer outra classe, que não esteja no mesmo pacote, acesse a coleção diretamente.

Você provavelmente deve fazer a mesma coisa para a coleção do outro lado.

E sobre o mapeamento do atributo inverse?

Pra você, e para o Java, uma associação bidirecional é simplesmente o fato de ajustar corretamente as referências de ambos os lados.

O Hibernate, entretanto não possui a informação necessária para adaptar corretamente os estados Insert e Update do SQL, e precisa de ajuda para manipular as propriedades das associações bidirecionais.

Fazer um lado da associação com o atributo inverse instrui o Hibernate para basicamente ignora-lo, considerando-o uma *cópia* do outro lado.

Isso é tudo o que é necessário para o Hibernate trabalhar com todas as possibilidades transformando um modelo de navegação bidirecional em esquema de banco de dados do SQL.

As regras que você deve lembrar são claras:

Todas as associações bidirecionais necessitam que um lado possua o atributo inverse.

Em uma associação de um-para-muitos, o lado de "muitos" deve conter o atributo inverse, já em uma associação de muitos-para-muitos você pode usar qualquer lado, não há diferença.

Exemplo simples de POJO

A maioria que aplicações de Java para empresas requerem uma classe persistente que represente funcionários.

Funcionarios.java

```
1 public class Funcionarios {
2
3     public Funcionarios(){
4     }
5     private String nome = "";
6     private int codigo = 0;
7     public void setNome(String nome){
8         this.nome = nome;
9     }
10    public String getNome(){
11        return this.nome;
12    }
13    public void setCodigo(int codigo){
14        this.codigo = codigo;
15    }
16    public int getCodigo(){
17        return this.codigo;
18    }
19
20 }
```

Regras Principais

Há quatro regras principais a se seguir:

1. Implementar um constructor sem argumentos

Funcionarios tem um constructor sem argumentos.

Todas a classes persistentes têm que ter um constructor default (que não precisa ser público) de forma que o Hibernate consiga criar uma nova instancia usando **Constructor.newInstance()**.

2. Garanta a existencia de uma propriedade identificadora (opcional)

Funcionarios tem uma propriedade chamada codigo.

Esta propriedade mapeia a coluna primary key na tabela.

A propriedade poderia ter qualquer nome, e seu tipo poderia ser qualquer tipo primitivo, qualquer "wrapper" para tipos primitivos, java.lang.Stringou java.util.Date. (Se sua tabela de banco de dados legado tiver chaves compostas, você pode até usar uma classe definida pelo usuario, com propriedades destes tipos)

A propriedade de identificador é estritamente opcional.
Você pode escolher não implementa-la e deixar que Hibernate gerencie internamente os identificadores dos objetos.
Sem dúvida, nós não recomendamos isso.

Na realidade, algumas funcionalidades só estão disponíveis a classes que declaram uma propriedade identificadora:

- Reassociação transitiva para objetos destacados (atualização de cascata ou merge em cascata)
- Session.saveOrUpdate()
- Session.merge()

Nós recomendamos que você declare propriedades identificadoras nomeadas de forma constantemente nas classes persistentes.
Nós também recomendamos que você use tipos que possam ser nulos (ou seja, tipos não primitivos).

3. Prefira classes que não sejam marcadas como final(opcional)

Uma apecto central do Hibernate, os proxies, dependem de que as classes persistentes não sejam finais, ou que sejam a implementação de uma interface que declara todos os métodos públicos.

Você pode persistir classes finaisque não implementam uma interface do Hibernate, mas você não poderá usar proxies para recuperação de associações lazy - o que limitará suas opções para a melhoria da perfomance.

Você deve evitar também declarar métodos como public finalnas classes que não sejam finais.

Se você quiser usar uma classe com um método public final, você deve desabilitar explicitamente o uso de proxies setando lazy="false".

4. Declare metodos accessores e modificadores para os campos persistentes(opcional)

Funcionarios declara métodos accessores para todos seus campos persistentes.

Muitas outras ferramentas de ORM persistem variáveis de instancias diretamente.

Nós acreditamos que é melhor prover uma indireção entre o schema relacional e a estrutura interna de classes.

Por default, o Hibernate persiste propriedades no estilo JavaBeans, e reconhecem nomes de métodos na forma getNome / setNome

Você pode trocar o acesso direto a campo por propriedades particulares, se precisar.

Propriedades não precisam ser declaradas com públicas

O Hibernate pode persistir uma propriedade com um par get / set protected ou private.

Implementando Herança

Uma subclasse também tem que obedecer a primeira e segunda regra.

Ela herda sua propriedade identificadora da superclasse, Funcionarios.

Motoristas.java

```
1 public class Motoristas extends Funcionarios {
2
3     public Motoristas(){
4     }
5
6     private String cnh = "";
7     public void setCnh(String cnh){
8         this.cnh = cnh;
9     }
10    public String getCnh(){
11        return this.cnh;
12    }
13
14 }
```

Avaliação da Aula

1- Criar a tabelas:

Consultores
idConsultor int, autonumeração e chave primária
nome VarChar(50) nascimento DateTime

Empresas
idEmpresa int, autonumeração e chave primária
empresa VarChar(50)
valorHora double

ConsultoresEmpresas
idConsultor int, chave primária
idEmpresa int, chave primária

2- Criar as classes e arquivos de mapeamento

3- Criar páginas jsp para:

3.1 Incluir consultores.

3.2 Incluir empresas.

3.3 Página para associar o consultor há uma empresa.

3.4 Listar empresas e consultores associados.
Esta página poderá remover uma associação