

Disciplina Regular 3

Desenvolvimento Web com Java EE

Graduação em Engenharia de Software - 2020

Etapa 4 Aula 2

Java Persistence API

Para acompanhar pelo Moodle você
deve estudar a etapa 6

Etapa 4 Aula 2 - Competências

- **Competências Trabalhadas Nesta Etapa**
 - **Compreender e utilizar o Java Persistence API.**
 - Dominar as técnicas de geração de código “bottom-up”.
 - Utilizar a sintaxe JQL para consultas.
 - **Implementar relacionamentos entre entidades.**
 - **Implementar transações declarativas.**

Etapa 4 Aula 2 - Checklist

- **Checklist DR3:**

- Ter revisado os slides e **os exercícios**, principalmente **JPA**.

- **Checklist PB:**

- Ter feito os primeiros commits do caso de uso “Manter Cadastro de Clientes” para o dia **11/5/2020**.
 - Inclusão, Consulta e Alteração.
- Ter iniciado os estudos sobre “**Sistemas de Recomendação**” e feito um esboço das especificações dos casos de uso “Manter Perfil de Cliente” e “Exibir Dashboard”.

Revisão da Aula Passada

Receita para o Spring MVC com JPA - 1 de 2

- **Criar o Projeto:**

- Criar o projeto web+maven normalmente: JEE 6 + JSE 8.
- Lembrar de colocar os projetos em um diretório "raso" (com nome curto) para facilitar os backups.


- **Configurar a Aplicação Web:**

- Criar a pasta WEB-INF e o web.xml.
- Copiar e colar.
 - applicationContext.xml
 - dispatcher-servlet.xml
- Configurar o Dispatcher Servlet no web.xml.

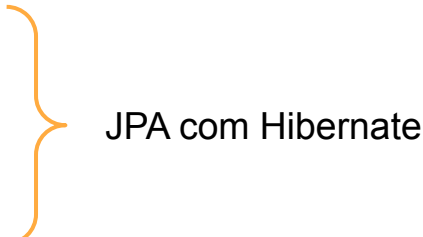


Spring_WEB-INF.zip

Receita para o Spring MVC com JPA - 2 de 2

- Dependências de Bibliotecas do Maven:
 - "org.springframework:**spring-webmvc**"
 - "org.hibernate:**hibernate-validator**"
 - "javax.servlet:**jstl**"

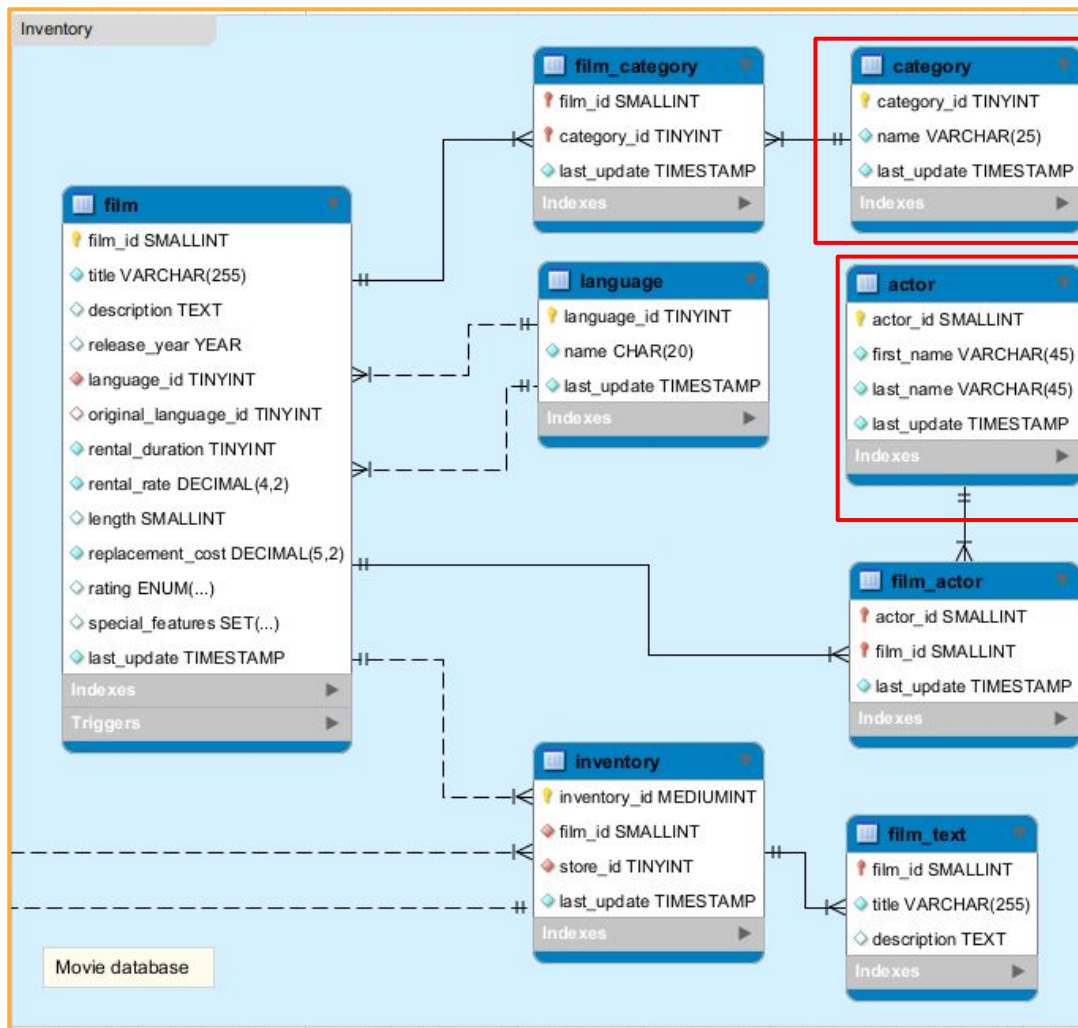
Spring MVC

 - "org.hibernate:**hibernate-entitymanager**"
 - "mysql:**mysql-connector-java**"
 - "org.springframework:**spring-orm**"

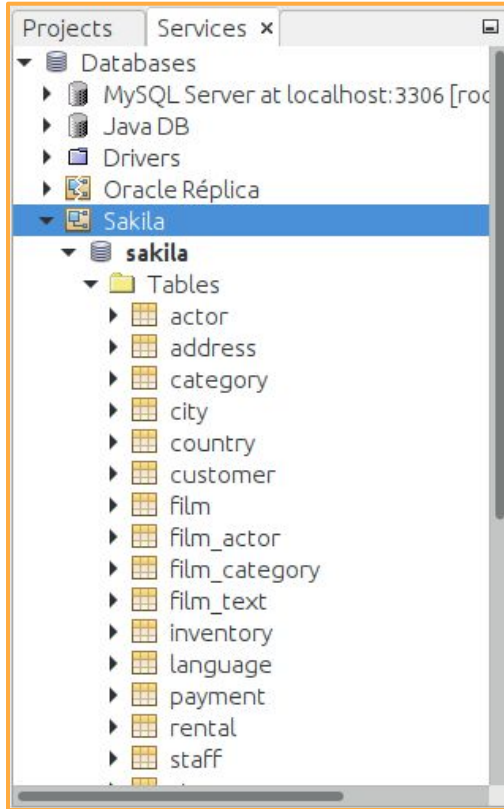
JPA com Hibernate



```
11 <!-- ===== -->
12 <!-- Configurações do JDBC DataSource -->
13 <!-- ===== -->
14 <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
15
16     <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
17     <property name="url" value="jdbc:mysql://localhost:3306/sakila" />
18     <property name="username" value="root" />
19     <property name="password" value="root" />
20
21 </bean>
22 <!-- ===== -->
23 <!-- Configurações do Gerenciador de Entidades -->
24 <!-- ===== -->
25 <bean id="entityManagerFactory"
26     class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
27     p:packagesToScan="br.edu.infnet"
28     p:dataSource-ref="dataSource">
29
30     <property name="jpaVendorAdapter">
31         <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
32             <!-- Aqui vão as configurações do Hibernate -->
33             <property name="generateDdl" value="false" />
34             <property name="showSql" value="true" />
35             <property name="databasePlatform" value="org.hibernate.dialect.MySQL5Dialect"/>
36         </bean>
37     </property>
38
39 </bean>
```

Engenharia Reversa



A screenshot of a Java code editor window titled 'Actor.java'. The code defines an entity for the 'actor' table in the 'sakila' database. It includes imports for JPA annotations and a public class 'Actor' that implements 'Serializable'. The class has private fields for 'actorId' (Short) and 'firstName' (String), both with 'GeneratedValue' and '@Basic' annotations. It also has a 'lastName' field (String) with a '@Basic' annotation. The code includes several named queries for finding actors by ID, first name, last name, and last update date.

```
1 package br.edu.infnet.actors;
2
3 import ...15 lines
4
5 @Entity
6 @Table(catalog = "sakila", schema = "", name="actor")
7 @NamedQueries({
8     @NamedQuery(name = "Actor.findAll", query = "SELECT a FROM Actor a"),
9     @NamedQuery(name = "Actor.findById", query = "SELECT a FROM Actor a WHERE a.actorId = :actorId"),
10    @NamedQuery(name = "Actor.findByFirstName", query = "SELECT a FROM Actor a WHERE a.firstName = :firstName"),
11    @NamedQuery(name = "Actor.findByLastName", query = "SELECT a FROM Actor a WHERE a.lastName = :lastName"),
12    @NamedQuery(name = "Actor.findByLastUpdate", query = "SELECT a FROM Actor a WHERE a.lastUpdate = :lastUpdate"))
13 public class Actor implements Serializable {
14
15     private static final long serialVersionUID = 1L;
16
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     @Basic(optional = false)
20     @Column(name = "actor_id", nullable = false)
21     private Short actorId;
22     @Basic(optional = false)
23     @NotNull
24     @Size(min = 1, max = 45)
25     @Column(name = "first_name", nullable = false, length = 45)
26     private String firstName;
27     @Basic(optional = false)
28     @NotNull
29     @Size(min = 1, max = 45)
30     @Column(name = "last_name", nullable = false, length = 45)
31     private String lastName;
32     @Basic(optional = false)
```

Relacionamentos

- Existem quatro tipos de multiplicidade e uma anotação para cada um:
 - **1 para 1**: anotação @OneToOne.
 - **1 para muitos**: anotação @OneToMany.
 - **Muitos para 1**: anotação @ManyToOne.
 - **Muitos para muitos**: anotação @ManyToMany.
- Possuem o atributo **mappedBy** que indica o campo da entidade que é “dona” do relacionamento (exceto ManyToOne).

Cascade & Fetch

- **Cascade:**

- Para OneToOne e OneToMany existe o atributo **cascade** para propagar uma operação para os filhos: REMOVE, PERSIST, MERGE, REFRESH, ALL.

- **Fetch:**

- Entidades relacionadas podem ser recuperadas de duas formas: objeto completo (**EAGER**) ou apenas o objeto pai (**LAZY**).
- No fetch do tipo EAGER, todos os objetos relacionados são recuperados do BD junto com o objeto pai.
- No fetch LAZY apenas o pai é recuperado e este é o default.

```
Country.java x
Source History
1 package br.edu.infnet.países;
2
3 import ...18 lines
21
22 @Entity
23 @Table(name = "country", catalog = "sakila", schema = "")
24 @NamedQueries({
25     @NamedQuery(name = "Country.findAll", query = "SELECT c FROM Country c"),
26     @NamedQuery(name = "Country.findAllWithCity", query = "SELECT c FROM Country c JOIN FETCH c.cityList"),
27     @NamedQuery(name = "Country.findById", query = "SELECT c FROM Country c WHERE c.countryId = :countryId"),
28     @NamedQuery(name = "Country.findByCountry", query = "SELECT c FROM Country c WHERE c.country = :country"),
29     @NamedQuery(name = "Country.findByLastUpdate", query = "SELECT c FROM Country c WHERE c.lastUpdate = :lastUpdate")
30 public class Country implements Serializable {
31
32     @Basic(optional = false)
33     @NotNull
34     @Size(min = 1, max = 50)
35     @Column(nullable = false, length = 50)
36     private String country;
37     @Basic(optional = false)
38     @NotNull
39     @Column(name = "last_update", nullable = false)
40     @Temporal(TemporalType.TIMESTAMP)
41     private Date lastUpdate;
42     @OneToMany(cascade = CascadeType.ALL, mappedBy = "countryId")
43     private List<City> cityList;
44
45     private static final long serialVersionUID = 1L;
46     @Id
```

Não disse nada,
então fetch é
lazy.

CountryRepository.java x

```

Source History
1 package br.edu.infnet.países;
2
3 import ...6 lines
9
10 @EnableTransactionManagement
11 @Repository
12 public class CountryRepository {
13
14     @PersistenceContext
15     private EntityManager em;
16
17     public List<Country> findAll() {
18
19         return em.createNamedQuery("Country.findAll").getResultList();
20     }
21
22     @Transactional
23     public void inserir(Country country) {
24
25         em.persist(country);
26     }
27 }

```

Root Cause

```

org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: br.edu.infnet.países.Country.cityList, could not initialize proxy - no Session
org.hibernate.collection.internal.AbstractPersistentCollection.throwLazyInitializationException(AbstractPersistentCollection.java:606)
org.hibernate.collection.internal.AbstractPersistentCollection.withTemporarySessionIfNeeded(AbstractPersistentCollection.java:218)
org.hibernate.collection.internal.AbstractPersistentCollection.readSize(AbstractPersistentCollection.java:162)
org.hibernate.collection.internal.PersistentBag.isEmpty(PersistentBag.java:376)
org.apache.el.parser.AstEmpty.getValue(AstEmpty.java:54)
org.apache.el.parser.AstNot.getValue(AstNot.java:43)
org.apache.el.ValueExpressionImpl.getValue(ValueExpressionImpl.java:190)
org.apache.jasper.runtime.PageContextImpl.proprietaryEvaluate(PageContextImpl.java:944)
org.apache.jsp.países.index_jsp._jspx_meth_c_005fif_005f1(index_jsp.java:388)
org.apache.jsp.países.index_jsp._jspx_meth_c_005fforEach_005f0(index_jsp.java:348)
org.apache.jsp.países.index_jsp._jspx_meth_c_005fwhen_005f0(index_jsp.java:292)
org.apache.jsp.países.index_jsp._jspx_meth_c_005fchoose_005f0(index_jsp.java:241)

```

CountryRepository.java x

Source

History



```

1  package br.edu.infnet.países;
2
3  + import ...6 lines
9
10 @EnableTransactionManagement
11 @Repository
12 public class CountryRepository {
13
14     @PersistenceContext
15     private EntityManager em;
16
17     public List<Country> findAll() {
18
19         return em.createNamedQuery("Country.findAllWithCity").getResultList();
20     }
21
22     @Transactional
23     public void inserir(Country country) {
24
25         em.persist(country);
26     }
27 }

```

Sakila - Países

[Inserir](#)

10 resultados por página

Pesquisar

ID ▲	País	Cidades
------	------	---------

1	Afghanistan	<table> <thead> <tr> <th>ID</th> <th>Cidade</th> </tr> </thead> <tbody> <tr> <td>251</td> <td>Kabul</td> </tr> </tbody> </table>	ID	Cidade	251	Kabul
ID	Cidade					
251	Kabul					

ID	Cidade
----	--------

59	Batna
----	-------