Java Server Faces

Usando JPA na Integração com Bancos de Dados

SoftBlue
www.softblue.com.br

Tópicos Abordados

- O que é e como surgiu a JPA
- Entidades
  - Trabalhando com entidades
  - A classe *EntityManager*
  - A linguagem JPQL
- O arquivo *persistence.xml*
- Pool de conexões e data sources
- Relacionamento entre entidades
  - Tipos de relacionamentos
  - Relacionamentos Eager e Lazy
- Transações

JPA – Java Persistence API

- A integração entre aplicações e bancos de dados relacionais é muito comum
- O problema é que aplicações e bancos de dados "falam línguas diferentes"

A especificação Java EE e o ORM

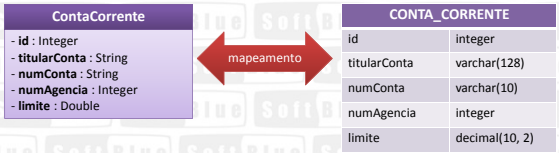
- Nas primeiras versões do Java EE (chamado de J2EE), a persistência de dados era feita pelos componentes denominados **entity beans**
  - Os entity beans eram complexos e limitados
- Em paralelo, surgiu o **Hibernate**
  - Permitia usar todas as facilidades da orientação a objetos a favor do ORM
  - Open source
- A **JPA** foi criada com base no Hibernate e foi incorporada na plataforma a partir do Java EE 5

Especificação da JPA

- A JPA é uma especificação
  - É apenas um documento com diretrizes para implementação da JPA
- As implementações da JPA são chamadas de **persistence providers**
  - EclipseLink (implementação de referência)
  - Hibernate
- Todas as implementações que seguem a especificação da JPA funcionam da mesma forma

Entidades

- Em orientação a objetos, chamamos as instâncias de classes de **objetos**
- No mundo ORM, os objetos que representam dados persistidos em tabelas do banco de dados são chamados de **entidades** (**entities**)



Criando Entidades

- Exemplo de implementação de uma entidade

```
@Entity
public class ContaCorrente {

    @Id
    @GeneratedValue
    private Integer id;

    private String titularConta;
    private String numConta;
    private Integer numAgencia;
    private Double limite;

    // getters e setters...
}
```

@Entity define a classe como sendo uma entidade

@Id define o ID da entidade

@GeneratedValue determina que o ID deve ser gerado automaticamente

Criando Entidades

- Existem outras configurações que podem ser utilizadas

```
//...

    @Column(nullable = false)
    private String titularConta;

    @Column(name = "num_conta", length = 10)
    private String numConta;
}

//...
```

@Column define propriedades da coluna de uma tabela

- Estes são apenas alguns exemplos

EntityManager

- A interface **EntityManager** é o ponto de entrada para o uso da JPA pelo programador
- Possui os métodos para interagir com entidades

Método	Descrição
persist()	Cria uma entidade
merge()	Atualiza uma entidade
remove()	Exclui uma entidade
find()	Busca uma entidade com base no seu ID
createQuery()	Cria um objeto *Query*, que permite procurar entidades de acordos com os critérios desejados

EntityManager

- Ao trabalhar com JSF, uma instância de *EntityManager* pode ser injetada no bean

```
@Named
@RequestScoped
public class MyBean implements Serializable {

  @PersistenceContext
  private EntityManager em;

  //...
}
```

@PersistenceContext injeta uma instância de *EntityManager*

JPQL

- A JPA define uma linguagem para busca de entidades
  - Esta linguagem tem bastante semelhança com o SQL, usado em bancos de dados relacionais
- **J**ava **P**ersistence **Q**uery **L**anguage
- A JPQL referencia apenas entidades
  - Não referencia tabelas ou colunas presentes no banco de dados

Exemplos de Queries em JPQL

- Buscar todas as contas correntes

```
SELECT c FROM ContaCorrente c
```

- Buscar contas correntes com limite superior a 1000

```
SELECT c FROM ContaCorrente c WHERE c.limite > 1000
```

- Buscar os nomes dos titulares de contas da agência 3456

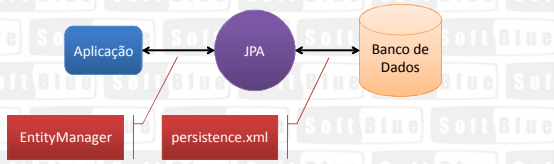
```
SELECT c.titularConta FROM ContaCorrente c
  WHERE c.numAgencia = 3456
```

- Buscar a quantidade de contas do titular *José Silva*

```
SELECT COUNT(c) FROM ContaCorrente c
  WHERE c.titularConta = 'José Silva'
```

O arquivo *persistence.xml*

- A JPA tem a responsabilidade de se comunicar com o banco de dados e fazer o gerenciamento das entidades



O arquivo *persistence.xml*

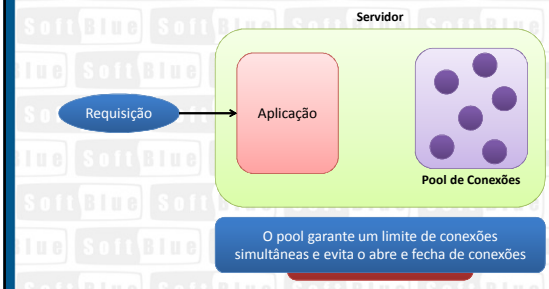
- O arquivo *persistence.xml* define como a JPA se comunica com o banco de dados
  - Normalmente é utilizada uma data source

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="appPU">
    <jta-data-source>jdbc/appds</jta-data-source>
  </persistence-unit>
</persistence>
```

Conexões com o Banco de Dados

- Em aplicações que rodam em um servidor, conexões com o banco de dados são normalmente gerenciadas pelo próprio servidor
  - Abrir conexões é um processo considerado caro computacionalmente
  - Existe um limite de conexões simultâneas que podem ficar ativas no banco de dados

Pool de Conexões

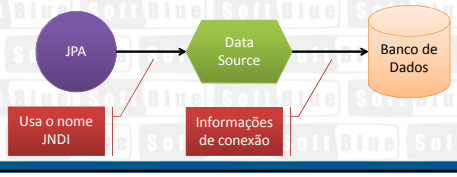


Pool de Conexões

- A configuração do pool de conexões varia de um servidor para outro
  - GlassFish, Tomcat, etc.
- É necessário consultar a documentação do servidor utilizado

Data Source

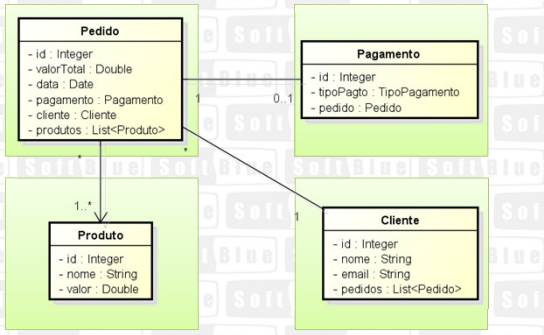
- A data source é uma fonte de dados
  - É configurada no servidor
  - Tem um pool de conexões associado
  - Tem um nome JNDI, que a identifica
    - Ex: **jdbc/appds**



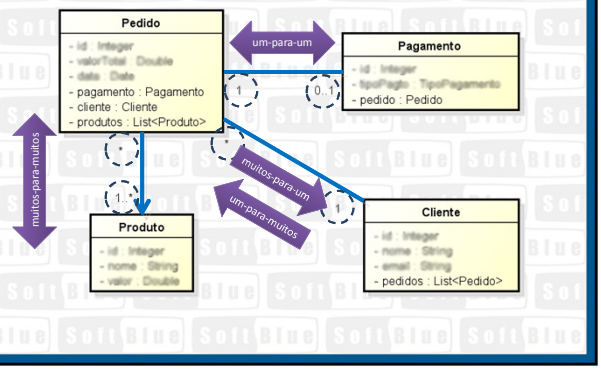
Relacionamentos em JPA

- O JPA também é capaz de trabalhar com relacionamentos entre entidades
  - Em orientação a objetos, um relacionamento existe quando um objeto da classe *A* possui um atributo que referencia um objeto *B*, de outra classe
  - No modelo relacional, um relacionamento existe quando uma tabela *A* referencia uma tabela *B* através de uma chave estrangeira (foreign key)

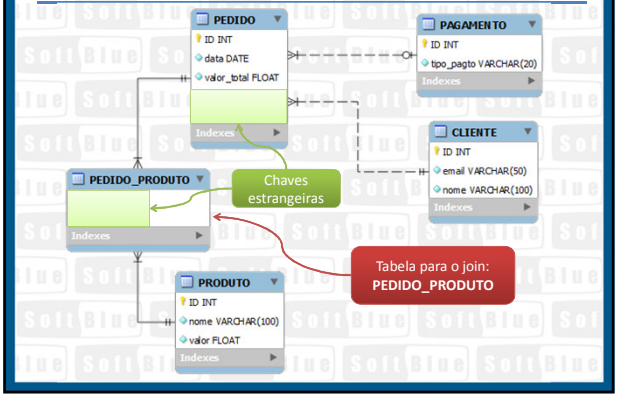
Exemplo na Orientação a Objetos



Exemplo na Orientação a Objetos



Exemplo no Modelo Relacional



Definindo Relacionamentos

- Os relacionamentos são definidos nas entidades através de anotações

Annotation	Tipo de Relacionamento
@OneToOne	Um-para-um
@OneToMany	Um-para-muitos
@ManyToOne	Muitos-para-um
@ManyToMany	Muitos-para-muitos

- Outras anotações utilizadas em relacionamentos

Annotation	Tipo de Relacionamento
@JoinColumn	Coluna para chave estrangeira
@JoinTable	Tabela auxiliar para o join

Definindo Relacionamentos

Pedido.java

```
@OneToOne
@JoinColumn(name = "pagamento_id")
private Pagamento pagamento;
```

```
@ManyToOne
@JoinColumn(name = "cliente_id", nullable = false)
private Cliente cliente;
```

```
@ManyToMany
@JoinTable(name = "PEDIDO_PRODUTO",
    joinColumns = @JoinColumn(name = "pedido_id"),
    inverseJoinColumns = @JoinColumn(name = "produto_id"))
private List<Produto> produtos;
```

Definindo Relacionamentos

Pagamento.java

```
@OneToOne(mappedBy = "pagamento")
private Pedido pedido;
```

Cliente.java

```
@OneToMany(mappedBy = "cliente")
private List<Pedido> pedidos;
```

Relacionamentos e a JPQL

- A linguagem JPQL também dá suporte a relacionamentos
- Os relacionamentos podem ser expressos de duas formas
  - Usando o ".", como acontece com qualquer propriedade de uma entidade
    - Possível em relacionamentos *um-para-um* e *muitos-para-um*
  - Usando o conceito de *join*, similar ao aplicado na linguagem SQL
    - [INNER] JOIN
    - LEFT [OUTER] JOIN
    - RIGHT [OUTER] JOIN

Exemplos


```
SELECT p FROM Pedido p WHERE p.cliente.nome = 'Pedro'
```

```
SELECT p FROM Pedido p INNER JOIN p.cliente c
  WHERE c.nome = 'Pedro'
```

```
SELECT p FROM Cliente c INNER JOIN c.pedidos p
  WHERE c.nome = 'Pedro'
```

Relacionamentos Eager e Lazy

- Quando uma entidade que possui relacionamentos é carregada, a JPA permite duas abordagens
  - Carregar automaticamente as entidades dos relacionamentos (**EAGER**)
  - Carregar os relacionamentos apenas quando eles forem necessários (**LAZY**)
  - A JPA assume um padrão
    - **@OneToOne** e **@ManyToOne** = **EAGER**
    - **@OneToMany** e **@ManyToMany** = **LAZY**

Exemplos

```
public class Pedido {
  @OneToOne(fetch = FetchType.EAGER)
  private Pagamento pagamento;
```

O pagamento será carregado automaticamente

```
public class Pedido {
  @OneToOne(fetch = FetchType.LAZY)
  private Pagamento pagamento;
```

O pagamento não será carregado automaticamente

Será carregado apenas quando for utilizado

Exemplos

```
public class Cliente {
  @OneToMany(fetch = FetchType.EAGER)
  private List<Pedido> pedidos;
```

Os pedidos serão carregados automaticamente

```
public class Cliente {
  @OneToMany(fetch = FetchType.LAZY)
  private List<Pedido> pedidos;
```

Os pedidos não serão carregados automaticamente

Serão carregados apenas quando forem utilizados

EAGER na JPQL

- Através da linguagem JPQL, é possível fazer com que relacionamentos definidos como *LAZY* se comportem como *EAGER*

```
SELECT c FROM Cliente c INNER JOIN c.pedidos p
  WHERE c.nome = 'Pedro'
```

Relacionamento LAZY

```
SELECT c FROM Cliente c INNER JOIN FETCH c.pedidos p
  WHERE c.nome = 'Pedro'
```

O uso do **FETCH** força o carregamento das entidades (modo EAGER)

Como Escolher entre Eager e Lazy

- Entre *EAGER* ou *LAZY*, não existe uma opção melhor ou pior
  - Vai depender de cada situação
- *EAGER*
  - Reduz o acesso ao banco de dados para leitura de dados
  - Ocupa mais memória
- *LAZY*
  - É preciso fazer vários acessos ao banco de dados para obter os dados conforme a necessidade
  - Ocupa menos memória

Transações

- O assunto transações é abrangente e complexo
- É importante saber que transações devem ser **atômicas**
  - Tudo executa ou nada executa
- Em JPA, todo código que modifica o banco de dados deve ser executado dentro de uma transação

Controle Transacional

- O controle das transações é feito através de um objeto **UserTransaction**
  - Pode ser injetado via anotação **@Resource**
- Três métodos são normalmente utilizados

Método	Descrição
begin()	Inicia a transação
commit()	Finaliza a transação com sucesso
rollback()	Finaliza a transação com falha

Demarcando Transações

```
public class Service {

    @Resource
    private UserTransaction ut;

    public void processar() {
        try {
            ut.begin();
            ...
            ...
            ...
            ut.commit();
        } catch (Exception e) {
            ut.rollback();
        }
    }
}
```

Gerenciador de transações

Executa numa mesma transação

Colocando em Prática...

Agora que você já aprendeu a teoria, acesse as vídeo-aulas práticas e pratique os assuntos abordados neste módulo!

Clique aqui para acessar as vídeo-aulas práticas