

O que é JPA?

- > Java Persistence API
- > Especificação padrão para mapeamento objeto relacional e gerenciamento de persistência da plataforma Java EE 5.0
- > Versão 1.0, faz parte da especificação JSR-220 (EJB 3.0)
- > Possui amplo suporte pela maioria dos grandes players do mercado: Apache, Oracle, BEA, JBoss, GlassFish

ORM: Mapeamento Objeto-Relacional

Objetos Java => ORM (Conversão) => JDBC (- Statements; - Result Sets) => Banco de Dados

- > Modelo OO vs Modelo Relacional
- > Classe = Tabela
- > Objeto = Linha
- > Atributo = Coluna
- > Associação = Chave Estrangeira
- > Mapeamento via XML ou Annotations

Funcionalidades da JPA

Padroniza Mapeamento Objeto-Relacional

Utiliza POJO's ao invés de Entity Beans

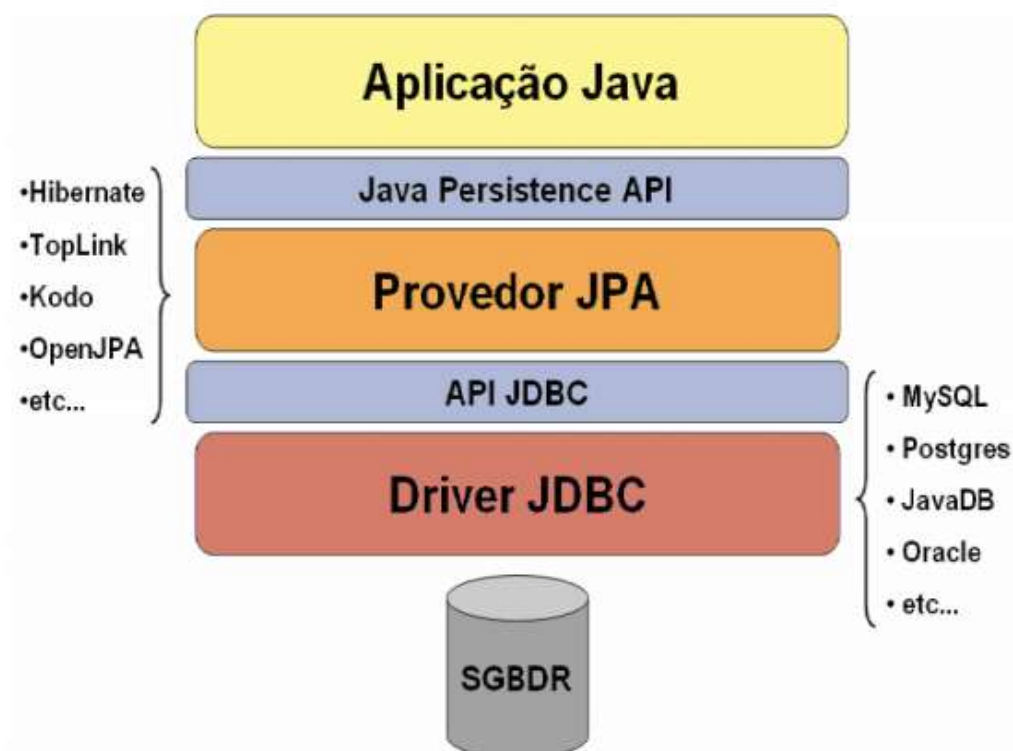
Pode ser usado com Java SE e Java EE

Suporta utilização de diferentes Providers

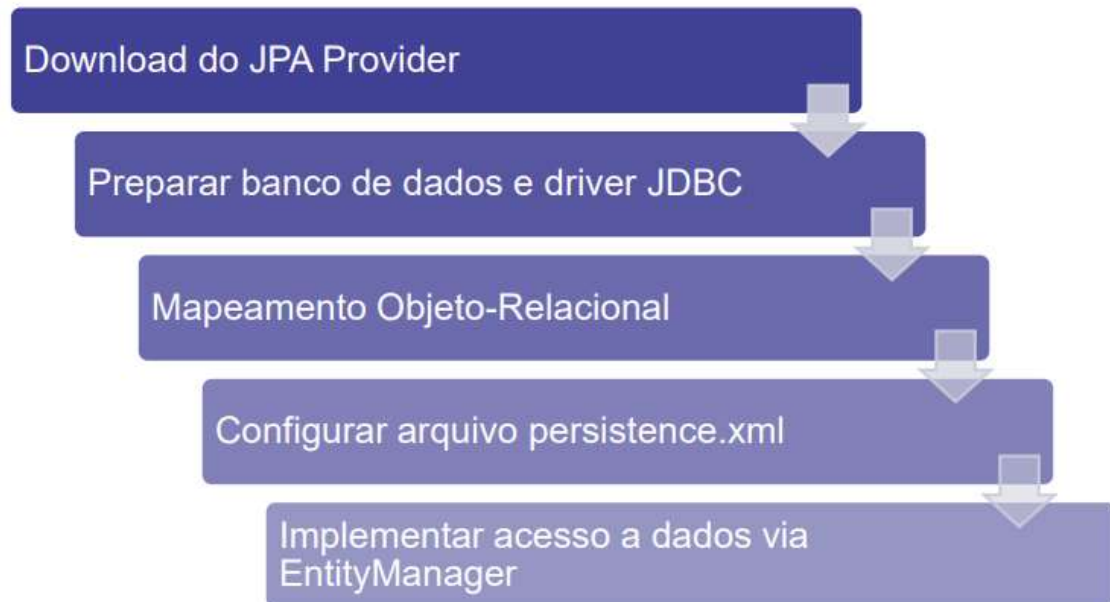
Possui uma linguagem de consulta estendida

Suporta herança, polimorfismo

Você pode utilizar seu framework preferido!



Passo a passo para utilização



JPA Providers

> Hibernate

- <https://hibernate.org/orm/>

> Toplink Essentials

- <https://oss.oracle.com/toplink-essentials-jpa.html>

> Open JPA

- <http://openjpa.apache.org/>

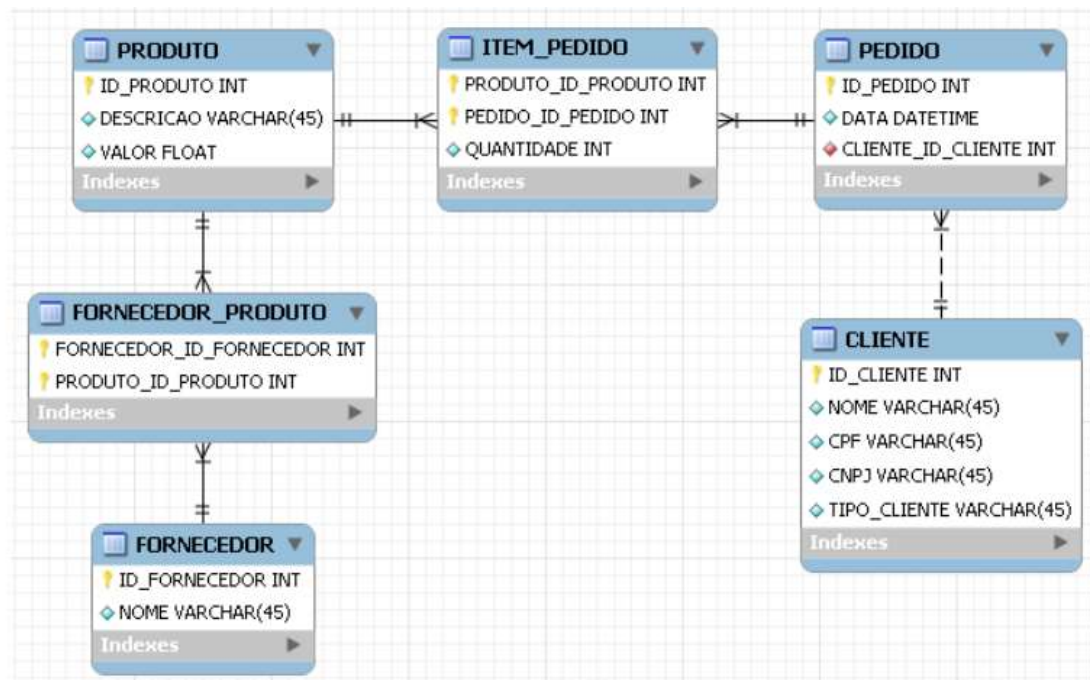
> Netbeans 8.2

– já vem com Hibernate, TopLink e EclipseLink!!

Banco de Dados e Driver JDBC

- > MySQL Community Server
- > <http://dev.mysql.com/downloads/mysql/5.0.html>
- > MySQL Connector/J 5.1
- > <http://dev.mysql.com/downloads/connector/j/5.1.html>
- > Ou banco de dados de sua preferência!

Modelo Relacional



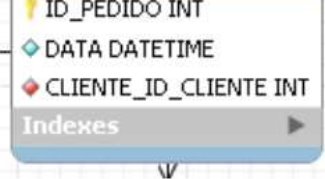
Mapeamento Objeto-Relacional

```
@Entity
@Table(name="PEDIDO")
public class Pedido implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_PEDIDO", nullable = false)
    private Integer idPedido;

    @Column(name = "DATA", nullable = false)
    @Temporal(TemporalType.TIMESTAMP)
    private Date data;

    @JoinColumn(name = "CLIENTE_ID_CLIENTE", referencedColumnName = "ID_CLIENTE")
    @ManyToOne(optional = false, fetch = FetchType.EAGER)
    private Cliente cliente;
```



The diagram shows a table named 'PEDIDO' with three columns: 'ID_PEDIDO' of type 'INT' with a primary key icon (yellow lightning bolt), 'DATA' of type 'DATETIME' with a diamond icon, and 'CLIENTE_ID_CLIENTE' of type 'INT' with a red diamond icon indicating a foreign key. Below the columns is a section labeled 'Indexes' with a right-pointing arrow.

Entidades

@Entity

- Especifica que uma classe é uma entidade
- Uma entidade é um objeto que pode ser persistido
- Representa uma tabela no banco de dados relacional

@Table

- Especifica nome da tabela no banco de dados

```
@Entity
@Table(name = "pedido", schema="jpa")
public class Pedido implements Serializable {
```

Atributos

@Column

- Mapeia um atributo ou uma propriedade (getter) a um campo do banco de dados
- Possui diversas opções de validação
- Lança javax.persistence.PersistenceException

```
@Column(name = "NOME", nullable = false, length = 45,  
        insertable = true, updatable = true, unique = false)  
private String nome;
```

Chave Primária Simples

@id

- Cada entidade precisa possuir uma chave primária
- Mapeia uma chave primária simples
- Chave pode ser gerada automaticamente: IDENTITY, AUTO, SEQUENCE, TABLE

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
@Column(name = "ID_PEDIDO", nullable = false)  
private Integer idPedido;
```

Chave Primária Composta

@Embeddable

- Define que uma classe pode fazer parte de uma entidade

```
@Embeddable
public class ItemPedidoPK implements Serializable {

    @Column(name = "PRODUTO_ID_PRODUTO", nullable = false)
    private int idProduto;

    @Column(name = "PEDIDO_ID_PEDIDO", nullable = false)
    private int idPedido;
```

Chave Primária Composta

@EmbeddedId

- Define uma propriedade que é embeddable como chave primária

```
@Entity
@Table(name = "item_pedido")
public class ItemPedido implements Serializable {

    @EmbeddedId
    protected ItemPedidoPK itemPedidoPK;
```

Relacionamentos

@ManyToOne

- Entidade Pedido

```
@JoinColumn(name = "CLIENTE_ID_CLIENTE", referencedColumnName = "ID_CLIENTE")  
@ManyToOne(optional = false, fetch = FetchType.EAGER)  
private Cliente cliente;
```

@OneToMany

- Entidade Cliente

```
@OneToMany(mappedBy = "cliente")  
private Collection<Pedido> pedidoCollection;
```

FetchType.EAGER

FetchType.LAZY

Obs: Lazy do inglês “preguiçoso, lento” e Eager que significa “ansioso, impaciente”

Enfim, o Lazy Loading faz com que determinados objetos não sejam carregados do banco até que você precise deles, ou seja, são carregados 'on demand' (apenas quando você solicitar explicitamente o carregamento destes).

Oposto ao Lazy Loading, o Eager Loading carrega os dados mesmo que você não vá utilizá-los, mas é óbvio que você só utilizará esta técnica se de fato você for precisar com muita frequência dos dados carregados.

@ManyToMany

- Entidade Produto

```
@JoinTable(name = "fornecedor_produto",
    joinColumns = {@JoinColumn(
        name = "PRODUTO_ID_PRODUTO",
        referencedColumnName = "ID_PRODUTO"}),
    inverseJoinColumns = {@JoinColumn(
        name = "FORNECEDOR_ID_FORNECEDOR",
        referencedColumnName = "ID_FORNECEDOR"}})

@ManyToMany
private Collection<Fornecedor> fornecedorCollection;
```

- Entidade Fornecedor

```
@ManyToMany(mappedBy = "fornecedorCollection")
private Collection<Produto> produtoCollection;
```

Operações em cascata

CascadeType:

- > PERSIST: Quando uma nova entidade é persistida, todas as entidades na coleção são persistidas
- > MERGE: Quando uma entidade desconectada é atualizada, todas as entidades na coleção são atualizadas
- > REMOVE: Quando uma entidade existente é removida, todas as entidades na coleção são removidas
- > ALL: Se aplicam todas as regras acima

Consultas

- > Java Persistence Query Language (JP-QL)
- > Define linguagem para consulta de entidades
- > Consultas baseadas nas entidades e suas propriedades, independente da modelagem física do banco de dados
- > Utiliza sintaxe próxima a SQL
- > Consultas estáticas (named queries) > Consultas dinâmicas

Consulta estática

- > Anotada na classe ou em arquivo XML separado

```
@Entity
@NamedQueries({
    @NamedQuery(name = "Pedido.findPedidoByIdCliente",
        query = "SELECT p FROM Pedido p WHERE cliente.idCliente = :idCliente")})
public class Pedido extends BaseEntity {
```

> Consulta dinâmica

```
String jpql = "SELECT c FROM Cliente c WHERE 1 = 1";
for (String paramName : filter.keySet()) {
    Object paramValue = filter.get(paramName);
    jpql += " AND " + paramName + " = " + paramValue + " ";
}
Query query = em.createQuery(jpql);
return query.getResultList();
```

Configurar Persistence.xml

- Hibernate

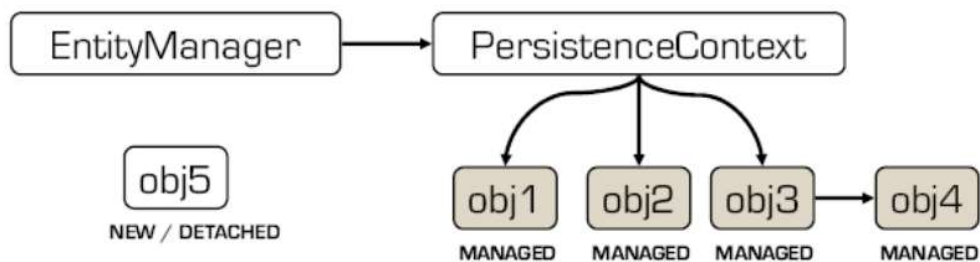
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www
  <persistence-unit name="tdc-floripa-jpa-projectPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>br.com.tdc.floripa.jpa.entity.Fornecedor</class>
    <class>br.com.tdc.floripa.jpa.entity.Pedido</class>
    <class>br.com.tdc.floripa.jpa.entity.Cliente</class>
    <class>br.com.tdc.floripa.jpa.entity.ClientePessoaFisica</class>
    <class>br.com.tdc.floripa.jpa.entity.ClientePessoaJuridica</class>
    <class>br.com.tdc.floripa.jpa.entity.Produto</class>
    <class>br.com.tdc.floripa.jpa.entity.ItemPedido</class>
    <properties>
      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
      <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/jpa"/>
      <property name="hibernate.connection.username" value="root"/>
      <property name="hibernate.connection.password" value=""/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

- TopLink

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www
  <persistence-unit name="tdc-floripa-jpa-projectPU" transaction-type="RESOURCE_LOCAL">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <class>br.com.tdc.floripa.jpa.entity.Fornecedor</class>
    <class>br.com.tdc.floripa.jpa.entity.Pedido</class>
    <class>br.com.tdc.floripa.jpa.entity.Cliente</class>
    <class>br.com.tdc.floripa.jpa.entity.ClientePessoaFisica</class>
    <class>br.com.tdc.floripa.jpa.entity.ClientePessoaJuridica</class>
    <class>br.com.tdc.floripa.jpa.entity.Produto</class>
    <class>br.com.tdc.floripa.jpa.entity.ItemPedido</class>
    <properties>
      <property name="toplink.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="toplink.jdbc.url" value="jdbc:mysql://localhost:3306/jpa"/>
      <property name="toplink.jdbc.user" value="root"/>
      <property name="toplink.jdbc.password" value=""/>
      <property name="toplink.logging.level" value="FINE"/>
    </properties>
  </persistence-unit>
</persistence>
```

Acesso a Dados

- > `javax.persistence.EntityManager`
- > Gerencia o ciclo de vida das entidades
- > NEW, MANAGED, DETACHED, REMOVED
- > Utilizado para criar e remover entidades, buscar entidades pela chave primária e fazer consultas
- > O conjunto de entidades que podem ser gerenciados por um `EntityManager` é definido dentro da Persistence Unit
- > `javax.persistence.PersistenceContext` > Conjunto de entidades associadas a um `EntityManager`



Acesso a Dados via JavaSE

Exemplo código.