# JPA resumo

## Configuração padrão persistence.xml

Primeiro passo é indicarmos para a jpa as configurações de conexão padrão com o banco de dados, fazemos isso pelo arquivo **persistence.xml** na pasta **META-INF**.

**Cabeçalho:**

<?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>

**Persistence-unit:**

Dentro de **persistence**, é preciso indicar para a JPA qual conjunto de configurações teremos, para que seja possível indicar qual banco de dados será utilizado.

Esse conjunto de configurações é chamado de **persistence-unit**, ao qual devemos dar um nome:

<persistence-unit name="default">

</persistence-unit>

**Properties:**

Dentro de properties são colocadas tags property contendo um atributo name indicando qual é a configuração que será feita, além de um atributo value com o conteúdo da configuração. Com isso, para o username, password e url de conexão, o XML fica similar a:

```
<?xml version="1.0" encoding="UTF-8"?>
        <persistence version="2.0" ... >
        <persistence-unit name="default">
<properties>
        <property name="javax.persistence.jdbc.url"
        value="jdbc:mysql://localhost/automoveis" />
        <property name="javax.persistence.jdbc.user"
        value="root" />
        <property name="javax.persistence.jdbc.password"
        value="password" />
        <property name="javax.persistence.jdbc.driver"
        value="com.mysql.jdbc.Driver" />

</properties>
</persistence-unit>
</persistence>
```

**Escolhendo o dialeto do banco de dados:**

O dialeto que será usado na aplicação pode ser indicado no persistence.xml por meio da propriedade hibernate.dialect, em que dizemos qual classe existente dentro do próprio Hibernate possui a implementação desse dialeto.

```
<property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLInnoDBDialect" />
```

A lista complete de dialetos pode ser encontrada em https://docs.jboss.org/hibernate/orm/4.1/javadocs/

**Criação das tabelas no banco de dados:**

O hibernate sempre se encarregara de converter as classes em tabelas podendo nos determinarmos os tipos de criação de tabelas:

**create**: O hibernate exclui e recria as tabelas, trabalhando sempre através de banco de dados vazios.

**create-drop:** As tabelas são criadas pelo hibernate e excluídas apenas ao final da execução.

**update:** Nada é excluído, apenas criado, ou seja, todos os dados são mantidos.

**validate:** Não cria nem exclui, apenas verifica se as entidades estão de acordo com as tabelas.

<property name="hibernate.hbm2ddl.auto"

value="update" />

**Imprimindo ações no console:**

Outro ponto interessante é que você pode fazer com que o Hibernate imprima

no console quais comandos SQLs estão sendo realizados no banco de dados. Para

isso, basta adicionar uma nova propriedade no persistence.xml, chamada

show_sql, com o valor true. Opcionalmente, você também pode usar a propriedade format_sql para que essa exibição saia formatada:

<property name="hibernate.show_sql" value="true" />

<property name="hibernate.format_sql" value="true" />

## Gravação de dados no banco de dados

**Persistindo classe no banco de dados:**

**EntityManagerFactory:**

Para persistimos um dado no banco de dados primeiro precisamos de um objeto da classe **EntityManagerFactory** utilizando o objeto da classe chamado

de **createEntityManagerFactory** da classe **Persistence** dentro desse método temos eu passar o nome da **persistence-unit** :

```
EntityManagerFactory emf = Persistence.
createEntityManagerFactory("default");
```

Esse método realiza diversas tarefas, entre elas, ler as anotações das entidades anotadas e criar o pool de conexões com o banco de dados. Por isso devemos criar apenas uma vez esse objeto pois ele pode levar a lentidão.

**EntityManager:**

Apartir da criação do objeto EntityManagerFactory e de ter acessado o método Persistence e ter dado o nome da persistence-unit, devemos instanciar um objeto da classe **EntityManager** e acessar o método **createEntityManager** da classe EntityManagerFactory:

```
EntityManager em = emf.createEntityManager();
```

**Persistir objeto:**

Para persistir o objeto utilizamos o método **persist** da classe **EntityManager**:

```
em.persist(objeto);
```

**EntityTransaction:**

Essa classe busca pedir a transação a EntityManager assim utilizamos o método **getTransaction**. Devemos dizer quando a transação começa e termina, através do método **begin** e **commit**.

Devemos sempre fechar as transações do EnityManager e EntityManagerFactory através do método **close.**

```
EntityTransaction tx = em.getTransaction();

tx.begin();

em.persist(objeto);

tx.commit();
```

**JPAUtil:**

Para garantirmos que a EntityManagerFactory seja criada apenas uma vez podemos criar uma classe que tenha um atributo estático para armazenar essa instância, e que ela seja uma constante, para que não seja criada outras vezes.

Podemos também fazer essa classe ter um método getEntityManager que devolva uma EntityManager, assim conseguimos ter acesso a ela sempre que precisarmos fazer as persistências:

```
public class JPAUtil {

private static final EntityManagerFactory emf =
Persistence.createEntityManagerFactory("default");

public static EntityManager getEntityManager() {
return emf.createEntityManager();
}
}
```

Com isso no mais so precisamos instanciar o método getEnityManager da classe JPAUtil no EntityManager:

```
public static void main(String[] args) {
EntityManager em = JPAUtil.getEntityManager();
}
```

## Consultas no banco de dados

**Consultas simples:**

Com o JPA é possível realizar consultas através do SQL porém os desenvolvedores a evitam e utilizam uma linguagem própria do JPA chamada **JPQL**

**Criando classe a ser persistida no banco de dados:**