



DIGITAL
INNOVATION
ONE

Java e o Banco de Dados

Java JPA Básico

Daniel Karam

Senior Software Developer

Objetivos da Aula

1. Entendendo o JPA

2. Implementações do JPA
(Hibernate e EclipseLink)

3. Linguagens de consulta
orientada a objetos

Requisitos Básicos

MySQL (SGBD) e noções de SQL

Java Development Kit (JDK) – 1.8 ou superior

IntelliJ 2019.2.3 IDE

Gradle 5.3.1 (Para baixar o Driver JDBC)

Materiais

Endereço no Github dos materiais que serão utilizados nessa aula:

https://github.com/danielkv7/digital-innovation-one/tree/master/Aula_JPA_basico

Parte 1: Entendendo o JPA

Java e o Banco de
Dados

Entendendo o JPA

Um problema de produtividade começou a ser notado no desenvolvimento de aplicações Web Java. Os desenvolvedores perceberam que a maior parte do tempo era gasto com queries SQL através do JDBC.

Um outro problema percebido era a mudança de paradigma. A programação orientada a objetos (ex: Java) é diferente do esquema entidade relacional (ex: SGBDs tradicionais), sendo necessário esquematizar dois modelos para um mesmo sistema.

Entendendo o JPA

Como solução para esses 2 problemas, foi proposto um modelo de mapeamento chamado **Mapeamento Objeto Relacional** (conhecido como **ORM**) para representar **tabelas** de um banco de dados relacional através de **classes** Java.

Exemplos de mapeamentos:

Tabela <---> Classe

Coluna <---> Atributo

Registro <---> Objeto

Entendendo o JPA

Para padronizar as interfaces das implementações ORM (Mapeamento Objeto Relacional) foi criada uma ESPECIFICAÇÃO oficial chamada JPA (ou Java Persistence API). Ela descreve como deve ser o comportamento dos frameworks Java ORM que desejarem IMPLEMENTAR a sua especificação.

Logo SOMENTE com a ESPECIFICAÇÃO JPA NÃO será possível executar operações entre a aplicação e o banco de dados.

Entendendo o JPA

Apesar de ser **SOMENTE** a especificação, o **JPA possui** algumas classes, interfaces e anotações que ajudam o desenvolvedor a abstrair o código.

Esses artefatos estão presentes no pacote **javax.persistence** que ajudam a manter o código **independente** da implementação utilizada. Lembrando que para persistir dados com JPA, **é preciso** escolher uma implementação que irá executar todo o trabalho (serão vistos na parte 2).

Entendendo o JPA

Entre os principais artefatos do JPA, podem ser destacados:

Anotação @Entity – Indica a aplicação que os OBJETOS da classe especificada serão persistidos no banco de dados. Também podem ser utilizadas outras anotações para auxiliar no mapeamento da classe, tais como: @id, @column, @table, @OneToMany e @ManyToOne.

Interface EntityManager – É utilizada para gerenciar o ciclo de vida das entidades. Os principais métodos utilizados são find, persist e remove.

Entendendo o JPA

As principais anotações utilizadas junto com a annotation @Entity são:

@Table – É uma annotation opcional. Por padrão o NOME da entidade é usado para realizar o mapeamento com o nome da **TABELA** do banco de dados. Essa annotation será necessária caso o nome da entidade seja diferente do nome da **tabela** no banco de dados.

@Column – É uma annotation opcional. Por padrão o ATRIBUTO da entidade é usado para realizar o mapeamento com o nome da **COLUNA** do banco de dados. Essa annotation será necessária caso os atributos da entidade sejam diferentes das **colunas** do banco de dados.

@Id – É OBRIGATÓRIO especificar ao menos uma ID para a entidade.

Entendendo o JPA

Também existem as annotations de relacionamento que são utilizadas para representar os relacionamentos entre TABELAS do banco de dados (através das chaves estrangeiras no banco de dados) em uma aplicação que esteja utilizando o **JPA**. As principais annotations são @ManyToMany, @ManyToOne, @OneToMany e @OneToOne.

Na aplicação utilizando JPA, é possível realizar relacionamento unidirecionais e bidirecionais. No unidirecional é possível chegar de uma instância A para uma instância B facilmente, porém o caminho contrário é dificultado. Na bidirecional, tanto do A para o B, quanto do B para o A o acesso é facilitado.

Entendendo o JPA

Nas annotations de relacionamento, a propriedade “fetch” exige atenção especial do desenvolvedor. Seus possíveis valores são eager (ansioso) ou lazy (preguiçoso). Suas características são:

Eager – A entidade mapeada com esse atributo SEMPRE será carregada na aplicação quando a entidade que está MAPEANDO for consultada, mesmo que nunca seja utilizada durante a execução da aplicação.

Lazy – A entidade mapeada com esse atributo SOMENTE será carregada na aplicação quando esta for EXPLICITAMENTE consultada pela entidade que está mapeando (É o mais aconselhável de usar caso não se saiba, em um primeiro momento, o real número de frequência de consultas).



Entendendo o JPA

Para persistir dados com as entidades mapeadas, é **OBRIGATÓRIO** iniciar uma transação. Para manipular transações, é necessário utilizar o seguinte método do **entityManager**:

getTransaction – Retorna uma **EntityTransaction**, sendo **obrigatório** o seu uso quando **utilizar algum método** que **realize alterações** no banco de dados. Pode utilizar os seguintes métodos:

begin – **Inicia** uma transação;

commit – Finaliza uma transação, **persistindo** todos os dados que foram modificados desde o início da transação;

rollback – Finaliza uma transação, **revertendo** todos os dados que foram modificados desde o início da transação.

Entendendo o JPA

Os principais métodos do entityManager para interagir com as entidades são:

find – Retorna a entidade que está persistida no banco de dados através da sua chave primária;

persist – Persiste a entidade no banco de dados (É necessário ter iniciado uma transação);

remove – Apaga a entidade do banco de dados (É necessário ter iniciado uma transação).

Entendendo o JPA

Para configurar uma aplicação JAVA para interagir com o banco de dados usando as especificações do JPA, será necessário configurar o arquivo persistence.xml.

Nele é possível especificar qual framework de implementação será utilizado (serão vistos na parte 2 do curso), quais classes serão mapeadas como ENTIDADES, URL de conexão, usuário, senha e driver (normalmente JDBC para BDs relacionais).



Entendendo o JPA

Passos para utilizar o JPA na sua aplicação:

1. Realizar download do Java Persistence API (JPA) e do driver JDBC para o BD MySQL. É possível baixar manualmente ou através do Gradle ou Maven
<https://mvnrepository.com/artifact/javax.persistence/javax.persistence-api>
<https://mvnrepository.com/artifact/mysql/mysql-connector-java>
1. Criar o arquivo persistence.xml e configurar os seguintes parâmetros: URL da string de conexão (driver, endereço do BD e nome do BD), usuário do BD, senha do BD, driver e classes que serão mapeadas para serem usadas pelo JPA.
1. Utilizar as annotations nas classes que serão mapeadas para uso do Hibernate.
2. Configurar o entityManager



Exercício final

1. Configure uma aplicação JPA de acordo com os passos explicados nos slides anteriores.

OBS: A IDE irá validar as annotations por que foi utilizada a API do JPA. Porém o código NÃO EXECUTARÁ! Pois NÃO foi utilizada nenhuma API de IMPLEMENTAÇÃO do JPA, e sim apenas a API com as ESPECIFICAÇÕES.

Parte 2: Implementações do JPA (Hibernate e EclipseLink)

Java e o Banco de Dados

Implementações do JPA (Hibernate e EclipseLink)

Lembrando que para utilizar o JPA É NECESSÁRIO utilizar alguma implementação, pois o JPA é apenas a ESPECIFICAÇÃO. Algumas das implementações mais conhecidas para o JPA são:

Hibernate é uma ferramenta **ORM** open source e é a líder de mercado, sendo a inspiração para a especificação Java Persistence API (JPA). O Hibernate nasceu **SEM JPA** e tinha sua própria implementação **ORM** (que ainda é possível usar), porém as versões atuais já possuem compatibilidade com a especificação JPA e são mais aconselháveis de usar do que a implementação nativa.

EclipseLink é um projeto open source de persistência da Eclipse Foundation. Ele é a implementação de referência do JPA, além de permitir desenvolvedores interagirem com vários serviços de data, incluindo banco de dados, web services, OXM (Object XML mapping), EIS (Enterprise Information Systems). Alguns padrões suportados pelo EclipseLink são: JPA, JAXB, JCA, SOD.



Implementações do JPA (Hibernate e EclipseLink)

É importante destacar que você pode encontrar sistemas com versões antigas do Hibernate utilizando as **APIs nativas** que foram desenvolvidas enquanto **NÃO** existia o **JPA**.

Mesmo o **JPA** sendo a **especificação oficial** para frameworks de implementação ORM, o **Hibernate AINDA** possui as suas **APIs nativas**. Elas são mais flexíveis porém mais complicadas de usar, portanto **é aconselhável** utilizar as **APIs** do **JPA** (caso não precise dessa flexibilidade).

Apenas como observação, as **APIs nativas** do **Hibernate** utilizam as classes “**SessionFactory**” e “**Session**” (no JPA são utilizados **EntityManagerFactory** e **EntityManager**). Porém, mesmo quando se utiliza o JPA com a implementação do Hibernate, na verdade são utilizadas as classes **SessionFactory** e **Session** de forma “envelopada” (wrapped)

Implementações do JPA (Hibernate e EclipseLink)

Para utilizar alguma implementação (Hibernate ou EclipseLink) com as especificações do **JPA**, basta seguir os seguintes passos:

1. Realizar o **download** da **API de implementação** desejada. É possível baixar manualmente ou através do Gradle ou Maven.
2. <https://mvnrepository.com/artifact/org.hibernate/hibernate-core/5.4.12.Final>
3. <https://mvnrepository.com/artifact/org.eclipse.persistence/eclipselink/2.7.6>
4. Modificar o arquivo **persistence.xml** configurando a tag **<provider>** indicando a classe da implementação que será utilizada.
5. É possível configurar parâmetros específicos de uma determinada implementação que foi escolhida no passo anterior (passo 2). Tais como o **dialeto do BD**, **log dos SQLs criados** e **automação dos comandos DDL** (ex: criar as tabelas no banco de dados quando a aplicação iniciar.)

Exercício final

1. Configure a aplicação desenvolvida com JPA na parte 1 para utilizar o Hibernate.
1. Configure a aplicação desenvolvida com JPA na parte 1 para utilizar o EclipseLink.

Parte 3: Linguagens de consulta orientada a objetos

Java e o Banco de Dados



Linguagens de consulta orientada a objetos

O **JPQL** (**Java Persistence Query Language**) é uma linguagem de consulta independente **orientada a objetos** definida pelo **JPA**.

JPQL é usado para realizar consultas no banco de dados. É inspirado no SQL (inclusive a sua sintaxe), porém ele interage com o banco de dados através das **entidades** do **JPA**, ao invés de interagir diretamente nas tabelas de banco de dados (como é no SQL).

Com o **JPQL** é possíveis utilizar as **propriedades de orientação a objetos** nas consultas realizadas no banco de dados, através das **entidades mapeadas**, tal como **herança**.



Linguagens de consulta orientada a objetos

Algumas vantagens ao utilizar o JPQL em relação aos métodos básicos de gestão de entidade do **EntityManager** são:

1. Operações de busca, atualização e remoção de entidades em MASSA, ao invés de realizar operações em apenas uma instância por vez através de chaves primárias (como nos métodos do entityManager);
2. Realizar consultas mais complexas;
3. Realizar funções de agregação.

Linguagens de consulta orientada a objetos

Algumas vantagens ao utilizar o JPQL em relação ao SQL são:

1. **NÃO** é necessário realizar os joins **explicitamente** entre entidades que estão com **annotations de relacionamento**, pois os joins são criados **automaticamente** durante uma consulta;
2. JPQL utiliza as funcionalidades de carregamento “**lazy** / **eager**” nos **relacionamento** entre entidades, aumentando a eficiência das consultas na aplicação.
3. As consultas podem ser armazenadas em cache para **melhor performance da aplicação**;



Linguagens de consulta orientada a objetos

Além do JPQL, existem outras linguagens para realizar consultas através dos frameworks ORM. Entre elas estão:

HQL - O Hibernate Query Language é uma linguagem de consulta orientada a objetos que realiza operações nas tabelas e colunas da base de dados através do Hibernate (através de classes e propriedades da orientação a objetos). Ela inspirou a criação do JPQL e para utilizá-la, é necessário utilizar as annotations nativas do Hibernate (session e sessionFactory).

EQL – O EclipseLink Query Language provê diversas extensões para a especificação padrão do JPQL. Essas extensões provêm acesso a funcionalidades padrões do SQL, além de funcionalidades específicas do EclipseLink.



Linguagens de consulta orientada a objetos

Existe uma alternativa a consultas **JPQL** a partir do **JPA 2.0** chamada **JPA Criteria API**, que é muito útil para construir **consultas dinâmicas**.

No **JPQL** as consultas só são verificadas no **momento da execução**, não sendo possível detectar erros de sintaxe na consulta durante a compilação. Já o **JPA Criteria API** consegue detectar esses erros no **momento de compilação**.

Essa funcionalidade se torna possível por que no **JPA Criteria API** as consultas são definidas como **instâncias de objetos Java** que representam elementos de consulta. Já as consultas **JPQL** são definidas apenas como **"string"**.



Linguagens de consulta orientada a objetos

No entanto, o JPA Criteria API é mais **complicado** de se utilizar do que o JPQL. Sendo assim, para consultas **estáticas simples**, é preferível utilizar o JPQL, enquanto que para consultas **dinâmicas** é preferível o JPA Criteria API.

Em relação a eficiência, tanto consultas JPQL quanto consultas JPA Criteria são **EQUIVALENTES** em **poder e eficiência**. Portanto, saber quando escolher um ou outro é um grande desafio para projetos de software.



Linguagens de consulta orientada a objetos

Para o **JPA Criteria API** verificar os possíveis erros em tempo de compilação, é necessário utilizar o **JPA Metamodel** para referenciar os **atributos** das entidades.

O **JPA Metamodel** provê a habilidade de examinar o modelo de persistência de um objeto para **consultar** os detalhes de uma **entidade JPA**. Para cada entidade, uma classes metamodelo é criada com o mesmo nome da classe, porém procedido pelo símbolo _ (*underscore*) e com os **atributos estáticos** que representam os campos de persistência.

Sem o **JPA Metamodel**, os atributos serão referenciados através de Strings, tendo como principal desvantagem o risco de ocorrer algum error em tempo de execução para o usuário final.



Linguagens de consulta orientada a objetos

Para usar o **JPQL** ou o **JPA Criteria API** é necessário ter um objeto da classe **EntityManager**, pois é através dos seus métodos **createQuery** (JPQL) e **getCriteriaBuilder** (JPA Criteria API) que se inicia a criação das consultas.

Para criar os **JPA Metamodel** de cada entidade será necessário adicionar o **JAR “hibernate-jpamodelgen”** através do Gradle, Maven ou manualmente. Esse JAR automatiza a criação de Metamodels (também existem outras organizações que oferecem esse tipo de solução).

É possível criar manualmente os **JPA Metamodels** de cada entidade que irão auxiliar na **validação das consultas** realizadas através do **JPA Criteria API**, porém isso seria trabalhoso demais. Por essa razão é mais fácil utilizar um gerador de Metamodels para automatizar esse processo.

Exercício final

1. **Crie** uma consulta **SQL** e **execute diretamente** no **Banco de Dados**;
2. **Realize** a mesma consulta realizada no passo 1, porém no **JPQL** e execute na sua aplicação **JPA**;
1. **Realize** a mesma consulta realizada no passo 1, porém com o **JPA Criteria API** e execute na sua aplicação **JPA**.

Contato

Linkedin -> <https://www.linkedin.com/in/daniel-kv/>