COMPUTAÇÃO UFCG

Banco de Dados 1

Introdução ao JPA com Hibernate

Java Persistence API (**JPA**) é uma API da linguagem Java que define uma interface padrão para frameworks de persistência de dados (ou frameworks ORM). A JPA define um conjunto de métodos capaz de realizar o mapeamento objeto-relacional (**ORM**), abstraindo uma série de detalhes do programador, dispensando a necessidade de programar as operações de persistência mais simples. Os objetos Java a serem persistidos no RDBMS são denominados beans de entidade.

O **Hibernate** é o principal Framework ORM da linguagem Java que segue a especificação da JPA. E utilizaremos ele neste roteiro.

Hoje iremos criar um mini projeto Java utilizando o Maven para gerenciamento de dependências que irá:

- Se conectar ao um banco de dados PostgreSQL com Hibernate
- Criar tabelas de forma automática com o mapeamento objeto-relacional
- Inserir, remover, alterar e recuperar dados para a aplicação Java de forma automática com o mapeamento objeto-relacional

Observações:

- Ao final da aula, o código estará disponível neste repositório: https://github.com/viniciuscva/crud_bd1
- O mini projeto consiste em apenas três classes: Aluno.java, AlunoDAO.java e Main.java
- O projeto será desenvolvido no Eclipse com o gerenciamento de dependências do Maven
- O arquivo dentro do caminho “src/main/resources/META-INF/persistence.xml” contém informações de acesso ao Banco de Dados. Este arquivo precisa ficar dentro deste caminho para que o Hibernate possa acessá-lo.

- Um backup dos arquivos JAR das libs necessárias pode ser solicitado caso não consiga baixar as dependências com o Maven.
- Sempre que achar necessário, consulte a documentação do Hibernate em https://hibernate.org/orm/documentation/5.6/

Ambiente de Execução

- Ferramentas necessárias
    - JDK 17
    - Eclipse 2023-06 com Maven
    - PostgreSQL 15
    - Algumas bibliotecas Java

OBS: A aplicação se conectará ao Postgres da Cloud da disciplina, portanto não é necessária a instalação do Postgres localmente.

Abaixo estão os métodos de ORM que você irá utilizar HOJE.

- *persist* - Persistir o objeto na tabela do Banco de dados;
- *find* - Recuperar o objeto da tabela para a aplicação Java usando a Primary key;
- *merge* - Alterar a linha referente a um objeto no Banco de dados;
- *remove* - Remover um objeto.
- *createQuery* - Criar uma consulta SQL mais personalizada

Anotações do JPA

Para realizar o correto mapeamento objeto-relacional (ORM), é necessário marcar as classes e métodos Java com algumas anotações do JPA, sendo as principais:

- *@Entity* - define que aquela classe deverá ser persistível no Banco
- *@Table* - permite definir o nome da tabela na qual irão persistir objetos Java
- *@Id* - define que aquele atributo será a Primary Key no Banco
- *@GeneratedValue* - define que um atributo (ex: PK) terá valor por autoincremento ou default
- *@Column* - permite definir um nome de coluna diferente da variável no programa
- *@ElementCollection* - define que um atributo representa uma lista/array e portanto irá criar uma tabela adicional para este atributo.
- *@OneToOne* - define a cardinalidade 1 para 1 em um relacionamento (ex: FK)
- *@ManyToOne* - define a cardinalidade Muitos para 1 em um relacionamento
- *@OneToMany* - define a cardinalidade 1 para Muitos
- *@ManyToMany* - define a cardinalidade Muitos para Muitos

OBS: Aqui trabalharemos apenas com as anotações @Entity, @Table, @Id e @Column.

Roteiro

1. Abra o Eclipse IDE e selecione “Criar Novo Projeto Maven Simples”.

2. Crie uma classe Aluno.java no pacote dominio, contendo os atributos matricula(String), nome(String) e email(String); Crie um construtor vazio e um construtor com atributos, também um método toString, e os Getters e Setters.

3. Crie uma classe Main.java e teste nela o instanciamento de alunos.

4. Agora vamos adicionar as dependências do nosso projeto. Verifique no Package Explorer do Eclipse, qual a versão do JRE, caso não for a 17, adicione as seguintes linhas no arquivo pom.xml e depois selecione Botão direito no projeto -> maven -> update project:

```
<properties>
  <maven.compiler.source>17</maven.compiler.source>
  <maven.compiler.target>17</maven.compiler.target>
</properties>
```

5. As dependências que utilizaremos são:
- https://mvnrepository.com/artifact/org.hibernate/hibernate-core/5.6.15.Final
- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager/5.6.15.Final
- https://mvnrepository.com/artifact/org.postgresql/postgresql/42.6.0

Edite o arquivo pom.xml para inserir os códigos dessas dependências

OBS: Caso esteja sem o Maven, pode obter as dependências baixando os arquivos JAR, e associando ao .classpath do projeto:

```
<classpathentry exported="true" kind="lib" path="lib/antlr-2.7.7.jar"/>
<classpathentry exported="true" kind="lib" path="lib/dom4j-1.6.1.jar"/>
<classpathentry exported="true" kind="lib"
path="lib/hibernate-commons-annotations-4.0.2.Final.jar"/>
<classpathentry exported="true" kind="lib" path="lib/hibernate-core-4.2.19.Final.jar"/>
<classpathentry exported="true" kind="lib"
path="lib/hibernate-entitymanager-4.2.19.Final.jar"/>
<classpathentry exported="true" kind="lib" path="lib/hibernate-jpa-2.0-api-1.0.1.Final.jar"/>
<classpathentry exported="true" kind="lib" path="lib/javassist-3.18.1-GA.jar"/>
<classpathentry exported="true" kind="lib" path="lib/jboss-logging-3.1.0.GA.jar"/>
<classpathentry exported="true" kind="lib"
path="lib/jboss-transaction-api_1.3_spec-2.0.0.Final.jar"/>
<classpathentry exported="true" kind="lib" path="lib/postgresql-42.2.19.jre7.jar"/>
```

Outra forma de fazer é: botão direito no projeto no eclipse -> Build Path -> Configure Build Path -> libraries -> add external JAR.

6. Crie a pasta META-INF dentro de src/main/resources e dentro da mesma, crie o arquivo persistence.xml, e cole nele o seguinte texto:

```
<?xml version="1.0" encoding="UTF-8" ?>
<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"
   version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">
   <persistence-unit name="crud" transaction-type="RESOURCE_LOCAL">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <properties>
         <property name="hibernate.dialect"
         value="org.hibernate.dialect.PostgreSQLDialect" />
         <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
         <property name="javax.persistence.jdbc.url"
value="jdbc:postgresql://150.165.15.11:5432/username_db" />
         <property name="javax.persistence.jdbc.user" value="username" />
         <property name="javax.persistence.jdbc.password" value="password" />

         <property name="hibernate.show_sql" value="true" />
         <property name="hibernate.format_sql" value="true" />
         <property name="hibernate.use_sql_comments" value="false" />
         <property name="hibernate.jdbc.wrap_result_sets" value="false" />
         <property name="hibernate.hibernate.cache.use_query_cache" value="true" />
         <property name="hibernate.hbm2ddl.auto" value="update" />
      </properties>
   </persistence-unit>
</persistence>
```

OBS: Personalize o preenchimento **username_db, username, password** com seu próprio banco de dados, usuário do BD e sua senha, respectivamente.

7. No arquivo Main.java, crie os objetos EntityManagerFactory e o EntityManager:

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("crud");
EntityManager em = emf.createEntityManager();
```

8. Edite o arquivo Aluno.java com a marcação @Entity (ele se tornará uma entidade monitorada (persistível) pelo hibernate), e seu atributo matricula com a marcação @Id do Hibernate (se tornará a Primary Key).

9. Instancie alunos no Main.java e salve (persista) no BD com o método persist

Exemplo de uso de PERSIST:

```
entityManager.getTransaction().begin()
entityManager.persist(aluno);
entityManager.getTransaction().commit()
```

10. Recupere alunos por matrícula

Exemplo de uso de FIND:

```
entityManager.find(Aluno.class, id);
```

11. Altere o email de um aluno

Exemplo de uso de MERGE:

```
aluno.setNome(“Novo nome”);
entityManager.getTransaction().begin()
entityManager.merge(aluno);
entityManager.getTransaction().commit()
```

12. Remova do Banco de dados um aluno existente

Exemplo de uso de REMOVE:

```
entityManager.getTransaction().begin()
entityManager.remove(aluno);
entityManager.getTransaction().commit()
```

13. Crie uma classe AlunoDAO.java e mova as chamadas ao JPA para essa classe, para padronizar o código. Crie na classe DAO os métodos findByMatricula, persist, update e removeByMatricula.

14. Crie um método para recuperar alunos pelo nome utilizando o método createQuery.

Exemplo de uso de createQuery:

```
Query query = entityManager.createQuery("FROM Aluno WHERE nome LIKE :nome");
query.setParameter("nome", "%"+nome+"%");
query.setMaxResults(1);
```

```
List<Aluno> listAluno = query.getResultList();
if(listAluno.isEmpty()) {
        System.out.println(“Nenhum aluno encontrado!”)l;
}
System.out.println(listAluno.get(0));
```

15. Altere o Main.java para criar requisições CRUD usando a classe AlunoDAO como forma de acesso ao BD.

Ao final da aula, tornarei público o repositório: https://github.com/viniciuscva/crud_bd1

Obrigado pela atenção, e bons estudos!

Referências

Hibernate 5.6 Documentation. https://hibernate.org/orm/documentation/5.6/

VÍDEO - JAVA com JPA #1 - Criando do zero projeto JPA com Hibernate (Canal Geek Dev). Disponível em: https://youtu.be/vtR3WAbC6IA?list=LL

VÍDEO - Introdução JPA e Hibernate (Canal DevSuperior). Disponível em: https://youtu.be/CAP1IPgeJkw?list=LL

VÍDEO - Criando entidades com chave estrangeira (Canal Descompila). Disponível em: https://www.youtube.com/watch?v=O03ialWcZJc

Advanced Spring Data JPA - Specifications and Querydsl: https://spring.io/blog/2011/04/26/advanced-spring-data-jpa-specifications-and-querydsl

Crud desenvolvido por Caio Libanio https://github.com/caiolibanio/crud_bd1