JPQL

1 – Introdução JPQL

Se você quer consultar um objeto e já sabe o identificador dele, pode usar os métodos find ou getReference de EntityManager. Caso o identificador seja desconhecido ou você quer consultar uma coleção de objetos, você precisará de uma query.

A JPQL (Java Persistence Query Language) é a linguagem de consulta padrão da JPA, que permite escrever consultas portáveis, que funcionam independente do SGBD. Esta linguagem de query usa uma sintaxe parecida com SQL, para selecionar objetos e valores de entidades e os relacionamentos entre elas.

2 – JPQL na Prática

Para entendermos na prática JPQL, iremos utilizar a nossa aplicação LivrosApp, criada com os tipos e os Livros. Lembrando que nos Livros temos o modelo com id, titulo, autor e tipo do livro. Para aumentar nossas possibilidade de consulta, iremos adicionar o ano de publicação do livro, ficando com o modelo da seguinte forma:

```java
@Entity
public class Livro {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    @Column(nullable = false, length = 100)
    private String titulo;

    @Column(nullable = false, length = 100)
    private String autor;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_tipo")
    private Tipo tipo;

    @Column(name = "ano_publicacao")
    private Integer anoPublicacao;
```

Agora precisaremos adicionar alguns Livros ao nosso Banco de Dados, para futuramente realizarmos nossas consultas JPQL.

```java
public class LivrosAppTest
{
    public static void main(String[] args) {
        //Cria uma fabricidade de entidades, utilizando o persistenceUnid, configurado em persistence.xml
        EntityManagerFactory fabricaGerenciadorEntidades =
                Persistence.createEntityManagerFactory( persistenceUnitName: "minhaConexao");
        //Cria um entityManager, que vai persistir os objetos marcados como @entity
        EntityManager gerenciadorEntidade = fabricaGerenciadorEntidades.createEntityManager();

        Tipo tipoRomance = gerenciadorEntidade.find(Tipo.class,  o: 1);
        Tipo tipoHistoria = gerenciadorEntidade.find(Tipo.class,  o: 2);
        Livro livro1 = new Livro( titulo: "Senhor dos Anéis",  autor: "J.R.R. Tolkien", tipoHistoria,  anoPublicacao: 1954);
        Livro livro2 = new Livro( titulo: "Harry Potter e a Pedra Filosofal",  autor: "J.K. RowLing", tipoHistoria,  anoPublicacao: 1997);
        Livro livro3 = new Livro( titulo: "A Culpa é das Estrelas",  autor: "John Green", tipoRomance,  anoPublicacao: 2012);

        gerenciadorEntidade.getTransaction().begin();
        gerenciadorEntidade.persist(livro1);
        gerenciadorEntidade.persist(livro2);
        gerenciadorEntidade.persist(livro3);
        gerenciadorEntidade.getTransaction().commit();

        gerenciadorEntidade.close();
        fabricaGerenciadorEntidades.close();

    }
}
```

Utilizando JPQL, podemos fazer consultas utilizando as entidades mapeadas via ORM (Mapeamento Objeto Relacional). Nossa primeira consulta iremos com o JPQL, buscar todos os livros existentes em nosso Banco de Dados.

Desta forma precisamos criar uma Query com o comando JPQL para busca as informações do banco de dados, conforme o mapeamento ORM adicionado em nosso modelo.

No caso temos a entidade Livro e adicionamos o alias l, então ao selecionar l, estamos selecionado um livro com todos os seus atributos.

Desta forma utilizando o getResultList, estaremos retornando uma lista de Livros para meusLivros.

No final temos uma expressão Lambda no foreach, apenas para demonstrar as informações selecionadas pela nossa JPQL.

Obs: Nos comandos demonstrados SQL em nossa aplicação, teremos um Select em Livros e um select em Tipos, serão demonstrados dois registros e depois teremos outro comando SQL em Tipos. Esta situação ocorre pelo tipo de fecth do nosso relacionamento, pois está definido como Lazy.

Desta forma os dados de tipos são selecionados no banco apenas quando necessários, como os dois primeiros livros são do mesmo tipo tem apenas uma consulta, porém no terceiro livro ele popula os dados do tipo, antes de demonstrar as informações no comando SQL.

```
Query minhaQuery = gerenciadorEntidade.createQuery( s: "select l from Livro l");
List<Livro> meusLivros = minhaQuery.getResultList();
meusLivros.forEach(livro -> System.out.println(livro));
```

Podemos ainda retornar uma lista apenas com os títulos dos livros, para isso precisamos especificar no comando JPQL a informação que precisamos.

```
Query minhaQuery = gerenciadorEntidade.createQuery( s: "select l.titulo from Livro l");
List<String> meusLivros = minhaQuery.getResultList();
meusLivros.forEach(livro -> System.out.println(livro));
```

Importante entendermos que a informação no comando JPQL, se refere aos atributos das entidades mapeadas e não ao banco de dados. Desta forma se precisarmos de uma lista dos anos de publicações, iremos utilizar a informação da entidade anoPublicacao e não do banco de dados ano_publicacao.

```
Query minhaQuery = gerenciadorEntidade.createQuery( s: "select l.anoPublicacao from Livro l");
List<Integer> meusLivros = minhaQuery.getResultList();
meusLivros.forEach(livro -> System.out.println(livro));
```

Podemos ainda adicionar condições para busca de informações. Por exemplo podemos selecionar apenas os livros que foram publicados depois de 2020, adicionado a condição where com os atributos JPQL.

```
Query minhaQuery = gerenciadorEntidade.createQuery( s: "select l from Livro l where l.anoPublicacao > 2000");
List<Livro> meusLivros = minhaQuery.getResultList();
meusLivros.forEach(livro -> System.out.println(livro));
```

```
Livro{id=7, titulo='A Culpa é das Estrelas', autor='John Green', tipo=Tipo{id=1, descricao='Romance Saltless'}}
```

Ou adicionado as informações do relacionamento. Neste caso selecionado os livros do tipo 2.

```
Query minhaQuery = gerenciadorEntidade.createQuery( s: "select l from Livro l where l.tipo.id = 2");
List<Livro> meusLivros = minhaQuery.getResultList();
meusLivros.forEach(livro -> System.out.println(livro));
```

```
Livro{id=5, titulo='Senhor dos Anéis', autor='J.R.R. Tolkien', tipo=Tipo{id=2, descricao='Historia'}}
Livro{id=6, titulo='Harry Potter e a Pedra Filosofal', autor='J.K. RowLing', tipo=Tipo{id=2, descricao='Historia'}}
```

O JPQL ainda permite que sejam adicionadas varáveis e as mesmas sejam alteradas em tempo de execução, antes da execução da Query. Neste caso com a mesma lógica que utilizamos nos comandos JDBC.

No exemplo temos as variáveis idTipo e ano no comando JPQL. Elas são substituídas, pelo setParameter.

```
Query minhaQuery = gerenciadorEntidade.createQuery( s: "select l from Livro l where l.tipo.id = :idTipo and l.anoPublicacao < :ano");
minhaQuery.setParameter( s: "idTipo",  o: 2);
minhaQuery.setParameter( s: "ano",  o: 1990);
List<Livro> meusLivros = minhaQuery.getResultList();
meusLivros.forEach(livro -> System.out.println(livro));
```

```
Livro{id=5, titulo='Senhor dos Anéis', autor='J.R.R. Tolkien', tipo=Tipo{id=2, descricao='Historia'}}
```

Para finalizar temos o conceito da Paginação. Esta técnica serve para definir uma limitação de registros a serem retornados pela consulta. O conceito é simples indicamos a partir de que registro os dados são retornados (setFirstResult) e depois indicamos quantos registros precisamos (setMaxResults).

Este conceito é muito utilizado em páginas que retornam listas, então definimos uma quantidade de registros a serem demonstrados e os novos registros serão retornados conforme a demanda do usuário. No exemplo são listados 10 registros a partir do primeiro, em uma nova consulta podermos solicitar mais 10 a partir do registro 11.

```
Query minhaQuery = gerenciadorEntidade.createQuery( s: "select l from Livro l");
minhaQuery.setFirstResult(0);
minhaQuery.setMaxResults(10);
List<Livro> meusLivros = minhaQuery.getResultList();
meusLivros.forEach(livro -> System.out.println(livro));
```

3 – Material Complementar

https://www.tutorialspoint.com/pg/jpa/jpa_jpql.htm

https://www.youtube.com/watch?v=R225whFQjNA

https://cafe.algaworks.com/livro-jpa-guia-definitivo/ (e-Book Gratuito)

4 – Exercícios

Em seu projeto do Cadastro de Pessoas, realize consultas JPQL. Buscando pessoas:

- Nome
- Idade
- Cidade
- Rua e Bairro