JPQL - Java Persistence Query Language

Introdução

- Estrutura de uma consulta JPQL:
  - SELECT ... FROM ... [WHERE ...] [GROUP BY ... [HAVING ...]] [ORDER BY ...]
  - DELETE FROM ... [WHERE ...]
  - UPDATE ... SET ... [WHERE ...]

JPQL Mínimo

- SELECT p FROM Pessoa AS p
- Versão JPQL de SELECT * FROM Pessoa (JPQL não aceita o *)
- A cláusula FROM declara uma ou mais variáveis de consulta (conhecidas como variáveis de identificação).
- Cada variável de consulta representa a iteração entre objetos no banco de dados.
- Uma variável de consulta que está vinculada a uma classe de entidade é chamada de variável de alcance.

JPQL Mínimo

- Variável de alcance:
  - define a iteração sobre todos os objetos armazenados no BD de uma classe de entidade vinculada e de suas classes relacionadas.
- Na consulta de exemplo, p é uma variável de alcance que está vinculada a classe de entidade Pessoa e define a iteração sobre todos os objetos da Classe Pessoa armazenados no BD

Tipos de Consultas

- Consultas são representadas em JPA 2 através de duas interfaces:
  - Query (JPA 1) e TypedQuery (JPA 2).
- TypedQuery estende Query.
- Em JPA 2 a Query é usada principalmente quando o tipo de retorno é desconhecido ou a Query retorna um resultado que é filho direto de Object.
- Quando é esperado um tipo mais específico de resultado normalmente se usa TypedQuery.

Exemplo

- Query:
  - Query q1 = em.createQuery("SELECT p FROM Pessoa p");
- TypedQuery:
  - TypedQuery<Pessoa> q2 = em.createQuery("SELECT p FROM Pessoa p", Pessoa.class);

Tipos de Consultas

- As consultas podem ser estáticas ou dinâmicas.
- Dinâmicas ocorrem dentro de uma string, como no exemplo anterior.
- A API Criteria provê uma alternativa para construir consultas dinâmicas, baseada em objetos Java que representam elementos de consulta (substituindo JPQL baseadas em strings).
- Estáticas ocorrem através de named queries.

Dinâmica - Exemplo

- Typed Query

```
public List<Venda> listar(){
  TypedQuery<Venda> query = em.createQuery("select v from Venda v", Venda.class);
   return query.getResultList();
}
```

Dinâmica - Exemplo

- Criteria

```
public List<Venda> listar(){
    CriteriaBuilder builder = em.getCriteriaBuilder();
    CriteriaQuery<Venda> criteria = builder.createQuery(Venda.class);
    criteria.from(Venda.class);
    return em.createQuery(criteria).getResultList();
}
```

Estática - Exemplo

- Named Query

```
@Entity
@NamedQuery(name="Venda.listar", query="SELECT v FROM Venda v")
public class Venda {
   ...
}
```

Estática - Exemplo

```
@Entity
@NamedQueries({
  @NamedQuery(name="Venda.listar", query="SELECT v FROM Venda v"),
  @NamedQuery(name="Venda.buscarPorNome", query="SELECT v FROM Venda v WHERE v.nome = :nome")
})
public class Venda {
  ...
}
```

Execução

- A Query define dois métodos para executar SELECT:
  - Query.getSingleResult() – para uso quanto é esperado exatamente um objeto de resultado.
  - Query.getResultList() – para uso geral.
- A Query define um método para DELETE e UPDATE:
  - Query.executeUpdate()

Execução

- A TypedQuery define dois métodos para executar SELECT:
  - TypedQuery.getSingleResult – para uso quanto é esperado exatamente um objeto de resultado.
  - TypedQuery.getResultList – para uso geral.

Exemplo

```
TypedQuery<Country> query = em.createQuery("SELECT c FROM Country c", Country.class);
List<Country> results = query.getResultList();

Query query = em.createQuery("SELECT c FROM Country c");
List results = query.getResultList()

for (Country c : results) {
    System.out.println(c.getName());
}
```

Exemplo

```
TypedQuery<Long> q = em.createQuery("SELECT COUNT(c) FROM Country c", Long.class);
long countryCount = q.getSingleResult();

Query q = em.createQuery("SELECT COUNT(c) FROM Country c");
long countryCount = (Long) q.getSingleResult();
```

Exemplo

```
int count = em.createQuery("DELETE FROM Country").executeUpdate();
int count = em.createQuery("UPDATE Country SET area = 0").executeUpdate();
```

- O executeUpdate retorna o número de objetos que foram excluídos ou atualizados.

Select

- JPQL pode retornar resultados diferentes de objetos de entidades.
- Ex: A consulta retorna uma String, ao invés de um país
  - SELECT c.name FROM Country AS c
- O código completo fica:

```
TypedQuery<String> query = em.createQuery( "SELECT c.name FROM Country AS c", String.class);
List<String> results = query.getResultList();
```

- É permitido o uso de atributos aninhados. Ex:
  - SELECT c.capital.name FROM Country AS c
- Para remover retornos duplicados, pode usar SELECT DISTINCT

Select

- É permitido buscar várias colunas: SELECT c.name, c.capital.name FROM Country AS c
- Código completo:

```
TypedQuery<Object[]> query = em.createQuery("SELECT c.name, c.capital.name FROM Country AS c", Object[].class);
  List<Object[]> results = query.getResultList();
  for (Object[] result : results) {
      System.out.println("Country: " + result[0] + ", Capital: " + result[1]);
  }
```

- A consulta retorna um array de Object.

Select

- Como alternativa ao array de Object, o JPA possui classes de resultados customizadas e expressões de construtores de resultados.
- O caminho completo da classe de resultados é especificado após o comando NEW:
  - SELECT NEW example.CountryAndCapital(c.name, c.capital.name) FROM Country AS c
- Para isso, a classe de resultado deve ter um construtor nesse formato.
- Pode ser tanto uma classe marcada com @Entity ou uma classe sem persistência definida.
- Se a classe não for marcada com @Entity, os objetos que retornam estarão DETACHED.

Select

```
public class CountryAndCapital {
    private String countryName;
    private String capitalName;

    public CountryAndCapital(String countryName, String capitalName) {
        this.countryName = countryName;
        this.capitalName = capitalName;
    }
}
```

Select

- Exemplo de executável:

```
String queryStr =
    "SELECT NEW example.CountryAndCapital(c.name, c.capital.name) " +
    "FROM Country AS c";
TypedQuery<CountryAndCapital> query =
    em.createQuery(queryStr, CountryAndCapital.class);
List<CountryAndCapital> results = query.getResultList();
```

From

- Após o FROM vai o nome da entidade
  - SELECT c FROM Country AS c
- O AS é opcional, podendo ser:
  - SELECT c FROM Country c
- Pode usar variáveis múltiplas
  - SELECT c1, c2 FROM Country c1, Country c2 WHERE c2 MEMBER OF c1.neighbors
  - Busca todos os países em c2 que fazem fronteira com c1
- Equivale ao uso de for aninhado em algoritmos.

From

- No exemplo SELECT c1, c2 FROM Country c1, Country c2 WHERE c2 MEMBER OF c1.neighbors:
- Dois FOR de algoritmos
- O FOR mais externo usa o c1 para iterar cada país cadastrado.
- O FOR mais interno usa o c2 para também iterar cada país cadastrado.
- O WHERE elimina todo par de países que não fazem fronteira entre si, retornando somente os países vizinhos.

From

- JOIN
  - SELECT c1, c2 FROM Country c1 INNER JOIN c1.neighbors c2
  - No exemplo, c1 é declarado como uma variável de alcance que itera sobre todos os objetos da entidade Country armazenados no BD.
  - c2 é declarada como uma variável de join que está vinculada ao caminho de c1.neighbors e itera sobre todos os elementos desta coleção
  - Faz o mesmo que o exemplo anterior, mas otimizado

From

- JOIN
  - SELECT c1, c2 FROM Country c1 INNER JOIN c1.neighbors c2
  - No exemplo, c1 é declarado como uma variável de alcance que itera sobre todos os objetos da entidade Country armazenados no BD.
  - c2 é declarada como uma variável de join que está vinculada a expressão de caminho c1.neighbors e itera sobre todos os elementos desta coleção
  - Isso faz o mesmo que o exemplo do slide anterior, mas otimizado

From

- JOIN
  - c2 pode ser tanto um objeto único quanto uma coleção,
  - Em INNER join é mais comum c2 ser uma coleção, porque:
  - Considere a consulta que retorna o país e o nome da sua capital:
    - SELECT c, c.capital.name FROM Country c
    - Isto é o mesmo que:
    - SELECT c1, c2.name FROM Country c1 INNER JOIN c1.capital c2
    - Somente quando c2 for um valor único (não uma coleção)!!!

From

- LEFT [OUTER] JOIN
  - Considerando o exemplo do INNER JOIN, mas retornando o nome do país e o nome da sua capital.
    - SELECT c1.name, c2.name FROM Country c1 INNER JOIN c1.capital c2
  - O FROM define a iteração entre os pares (country, capital).
  - Um país sem capital (ex.: República de Nauru, que não possui uma capital oficial) não faz parte de nenhum par de iteração e é excluída do resultado.
  - O LEFT JOIN retorna também o par (Nauru, null).
- Não existe RIGHT JOIN em JPQL!!!

From

- [LEFT [OUTER] | INNER] JOIN
  - Usado para buscar dados parciais de uma lista de objetos quando ativado o modo LAZY.
  - Ex: SELECT c FROM Country c JOIN FETCH c.capital
  - Retorna imediatamente cada objeto da entidade Country com a capital no modo EAGER, e o restante dos dados no modo LAZY.

Where

- Exemplo
  - SELECT c FROM Country c WHERE c.population > :p
  - No código ficaria assim:
  - String queryStr = “SELECT c FROM Country c WHERE c.population > :p”
  - TypedQuery<Country> query = em.createQuery(queryStr, Country.class);
  - query.setParameter(“p", 1000000);
  - List<Country> results = query.getResultList();

Where

- Como parâmetro podem ir tipos simples, objetos ou coleções
- Exemplo
  - SELECT c, l FROM Country c JOIN c.languages l WHERE c.population > :p AND l IN :languages
  - Com isso, os parâmetros poderiam ser assim:
  - query.setParameter("p", 1000000);
  - List<String> langs = new ArrayList<String>();
  - langs.add("Portuguese"); langs.add("Spanish");
  - query.setParameter("l", langs);
  - List<Object[]> results = query.getResultList();

Where

- Para acessar o resultado:

```
for (Object[] result : results) {
    System.out.println("Country: " + result[0] + ", Language: " + result[1]);
}
```

Where

- Existem 2 formas de acessar listas:
  - Object[] e Classes de resultado customizadas
- Object[] necessita saber o tamanho do array
- Classe de resultado customizada necessita do comando NEW
- SELECT NEW CountryAndLanguage(c, l) FROM Country c JOIN c.languages l WHERE c.population > :p AND l IN :languages
- Para isso deve existir uma classe com um construtor com o mesmo nome e parâmetros da consulta.

Where

```
public class CountryAndLanguage {
    public Country country;
    public String language;

    public CountryAndLanguage(Country country, String language) {
        this.country = country;
        this.language = language;
    }
}
```

GROUP BY e HAVING

- Permite agrupar itens por funções
  - COUNT, SUM, AVG, MIN, MAX
  - Ex: SELECT SUBSTRING(c.name, 1, 1), COUNT(c), COUNT(DISTINCT c.currency) FROM Country c GROUP BY SUBSTRING(c.name, 1, 1);
  - Retorna, a partir da primeira letra do país, quantos países começam pela letra, e quantas moedas são usadas pelos países dessa letra.
  - O DISTINCT é usado para evitar a contagem repetida de moedas.
- Somente o COUNT pode ser aplicado diretamente em objetos (Ex: COUNT(c)).
- As outras funções devem ser aplicadas em atributos (Ex: SUM(c.population)

GROUP BY e HAVING

- Having é usado para filtrar o group by, da mesma forma que o where é usado para filtrar o from.
- SELECT c.currency, SUM(c.population) FROM Country c WHERE 'Europe' MEMBER OF c.continents GROUP BY c.currency HAVING COUNT(c) > 1
- Somente pode entrar na comparação termos relacionados ao GROUP BY.

Agregações globais

- Quando somente as funções de agregações são retornadas em uma consulta, o group by é implícito.
- Ex: SELECT SUM(c.population), AVG(c.population) FROM Country c WHERE 'English' MEMBER OF c.languages
- Todos os objetos e seus atributos ficam inacessíveis.

ORDER BY

- Mesma estrutura de SQL, vindo ao final da consulta.
- Ex:
  - SELECT c.name FROM Country c WHERE c.population > 1000000 ORDER BY c.name
  - SELECT c.currency, c.name FROM Country c ORDER BY c.currency, c.name
  - SELECT c.name FROM Country c ORDER BY c.name ASC
  - SELECT c.name FROM Country c ORDER BY c.name DESC

Parâmetros

- Parâmetros nominais
  - :variável
  - Para definir um valor para a variável:
  - query.setParameter(“variável", valor);
  - Ex:
  - SELECT c FROM Country c WHERE c.population > :p
  - query.setParameter(“p", 1000000);

Parâmetros

- Parâmetros ordinais
  - Define uma ordem para os parâmetros
  - Representado por ?numero
  - Ex: "SELECT c FROM Country c WHERE c.name = ?1“
  - query.setParameter(1, name)

Parâmetros especiais

- query.setParameter("date", new java.util.Date(), TemporalType.DATE);
- Query.setParameter(date, new java.util.Calendar(), TemporalType.DATE)