Tutorial: Principais *Query Methods* no Spring Data JPA

Este tutorial apresenta os principais **Query Methods** utilizados no Spring Data JPA, mantendo o exemplo das classes `Client` e `Address`. Vamos mostrar como consultas personalizadas facilitam o acesso a dados específicos em entidades, tudo sem a necessidade de SQL explícito, graças às convenções de nomeação do Spring Data JPA.

1. Objetivo

Neste tutorial, vamos explorar os Query Methods principais do Spring Data JPA:

- Consultar dados específicos de `Client`.
- Buscar informações de `Address`.
- Utilizar as convenções de nomeação para consultas sem SQL explícito.

2. Estrutura do Projeto

Vamos manter o modelo já utilizado, com as seguintes classes e repositórios:

- **Entidades**: `Client` e `Address`
- **Repositórios**: `ClientRepository` e `AddressRepository`

---

3. Palavras-chave suportadas dentro de nomes de métodos

Palavra-chave	Amostra	Trecho JPQL
Distinct	findDistinctByLastnameAndFirstname	select distinct … where x.lastname = ?1 and x.firstname = ?2
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname, findByFirstnameIs,findByFirstnameEquals	… where x.firstname = ?1
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull,Null	findByAge(Is)Null	… where x.age is null

Palavra-chave	Amostra	Trecho JPQL
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age is not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1(parâmetro vinculado com %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1(parâmetro vinculado com prefixado %)
Containing	findByFirstnameContaining	… where x.firstname like ?1(parâmetro vinculado encapsulado em %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstname) = UPPER(?1)

4. Criação de `ClientRepository` com Query Methods Básicos

A interface `ClientRepository` herda métodos CRUD de `JpaRepository` (como `findAll()`, `findById()`, `save()`). Vamos adicionar Query Methods específicos para `Client` com convenções de nomenclatura:

```
public interface ClientRepository extends JpaRepository<Client, Long> {

    // Buscar por nome exato
    List<Client> findByName(String name);

    // Buscar clientes com renda maior que um valor específico
    List<Client> findByIncomeGreaterThan(Double income);

    // Buscar clientes por número exato de filhos
    List<Client> findByChildren(Integer children);

    // Buscar cliente específico por CPF
    Optional<Client> findByCpf(String cpf);

    // Buscar clientes nascidos após uma data
    List<Client> findByBirthDateAfter(Instant birthDate);

    // Buscar clientes pelo nome, ignorando maiúsculas e minúsculas
```

```
    List<Client> findByNameIgnoreCase(String name);

    // Buscar clientes com renda entre valores específicos
    List<Client> findByIncomeBetween(Double minIncome, Double maxIncome);

    // Buscar clientes com pelo menos um número mínimo de filhos
    List<Client> findByChildrenGreaterThanEqual(Integer minChildren);

    // Buscar clientes por termo em qualquer parte do nome
    List<Client> findByNameContaining(String term);

    // Buscar clientes por nomes com um prefixo específico
    List<Client> findByNameStartingWith(String prefix);
}
```

Explicação dos Principais Query Methods em `ClientRepository`

- **Filtros numéricos** (`findByIncomeGreaterThan`, `findByChildrenGreaterThanEqual`) e de intervalo (`findByIncomeBetween`).
- **Filtros de data** (`findByBirthDateAfter`).
- **Filtros de texto** (`findByNameContaining`, `findByNameStartingWith`).

Esses métodos permitem acessar os dados dos clientes de maneira personalizada, com base em critérios específicos.

5. Criação de `AddressRepository` com Query Methods

Para `AddressRepository`, criaremos métodos para consultar endereços por localidade e atributos específicos:

```
public interface AddressRepository extends JpaRepository<Address, Long> {

    // Buscar endereços por cidade
    List<Address> findByCity(String city);

    // Buscar endereços por estado
    List<Address> findByState(String state);

    // Buscar endereços por cidade e estado combinados
    List<Address> findByCityAndState(String city, String state);

    // Buscar endereços por código postal (CEP)
    List<Address> findByZipCode(String zipCode);

    // Buscar endereços por rua contendo um termo específico
    List<Address> findByStreetContaining(String term);
}
```

Explicação dos Principais Query Methods em `AddressRepository`

- **Filtragem por localização**: Métodos como `findByCity` e `findByState` permitem consultas por cidade e estado específicos.
- **Busca por atributos**: `findByZipCode` para filtrar endereços por código postal e `findByStreetContaining` para filtrar por termos contidos no nome da rua.

6. Testando os Query Methods em `Main`

Para testar nossos métodos de consulta, implementamos a classe `ApplicationMain`, executando as operações no método `main`:

```
@SpringBootApplication
public class ApplicationMain implements CommandLineRunner {

    @Autowired
    private ClientRepository clientRepository;

    @Autowired
    private AddressRepository addressRepository;

    public static void main(String[] args) {
        SpringApplication.run(ApplicationMain.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        // Criando instâncias de Client e Address
            Client client1 = new Client(null, "John Doe", "12345678900", 5000.0,
Instant.parse("1990-01-01T00:00:00Z"), 2);
            Client client2 = new Client(null, "Jane Doe", "98765432100", 6000.0,
Instant.parse("1992-02-02T00:00:00Z"), 1);
            clientRepository.save(client1);
            clientRepository.save(client2);

            Address address1 = new Address(null, "123 Main St", "New York",
"NY", "10001", client1);
            Address address2 = new Address(null, "456 Elm St", "Los Angeles",
"CA", "90001", client2);
            addressRepository.save(address1);
            addressRepository.save(address2);

        // Testando Query Methods
            List<Client>   clientsByName   =   clientRepository.findByName("John
Doe");
            System.out.println("Clientes   com   nome   'John   Doe':   "   +
clientsByName);
            List<Client>                clientsWithHighIncome                =
clientRepository.findByIncomeGreaterThan(5500.0);
            System.out.println("Clientes   com   renda   maior   que   5500:   "   +
clientsWithHighIncome);
            List<Client>               clientsWithMinChildren                =
clientRepository.findByChildrenGreaterThanEqual(1);
            System.out.println("Clientes   com   pelo   menos   1   filho:   "   +
clientsWithMinChildren);
            List<Address>  addressesInCity  =  addressRepository.findByCity("New
York");
            System.out.println("Endereços   na   cidade   de   Nova   York:   "   +
addressesInCity);
            List<Address>                 addressesByState                  =
addressRepository.findByState("CA");
            System.out.println("Endereços   no   estado   da   Califórnia:   "   +
addressesByState);
    }
}
```