

Logo da Orange Talents

# Métodos que geram queries no Micronaut Data

## Antes de começar

Para complementar o curso, recomendamos estudar a documentação oficial do Micronaut sobre [Writing Queries](https://micronaut-projects.github.io/micronaut-data/latest/guide/#querying) caso você não tenho estudado ainda. Nela é possível ter uma idéia mais clara do poder e das possibilidades oferecidas pelo framework.

Outro detalhe importante é que quase tudo que aprendemos no curso da ALURA de [Spring Data JPA: Repositórios, Consultas, Projeções e Specifications](https://www.alura.com.br/curso-online-spring-data-jpa) pode ser aproveitado no aprendizado com Micronaut Data.

## Cenário:

Imagine que nosso time tenha que implementar uma feature em um aplicativo mobile de cartão de crédito na qual precisará gerar um relatório das transações de pagamento feitas nos últimos 30 dias pelo cliente (usuário) . Para que isso seja possível, o time mobile nos solicitou uma API REST para que eles possam consumir os dados das transações.

Parte dessa API REST já foi implementada por outro desenvolvedor(a) do nosso time, mas ainda falta escrever o código responsável por carregar as transações do banco de dados. Esta consulta deve utilizar o repository do Micronaut existente no projeto na qual possui o código abaixo juntamente com a entidade da JPA que representa a transação:

@Repository  
interface TransacaoRepository : JpaRepository<Transacao, Long> {  
  
 // outros métodos implementados pelo time  
}  
  
@Entity  
class Transacao(  
 val descricao: String,   
 val valor: BigDecimal,   
 val clienteId: Long,   
 @Enumerated(STRING)  
 val status: StatusDaTransacao) {  
  
 @Id  
 @GeneratedValue  
 val id: Long? = null  
  
 val criadaEm: LocalDateTime = LocalDateTime.now()  
  
 // varios outros atributos  
}

Como o relatório será exibido diretamente no aplicativo mobile o time mobile não precisa de muitas informações, precisando apenas de 3 campos da transação: descrição, valor e data de criação; além disso, as transações precisam vir ordenadas pela data de criação de tal forma que as transações mais recentes sejam listadas primeiro. Nosso tech lead nos lembrou que alguns clientes fazem dezenas ou mesmo centenas de pagamentos via cartão de crédito diariamente, então pode ser comum que alguns clientes possam chegar a milhares de transações por mês.

Dado que a implementação da API REST não é uma preocupação para gente nesse momento, como você implementaria o código para consultar as transações de um cliente no banco de dados via repository do Micronaut?

## O que seria bom ver nessa resposta?

O que código e sintaxe da assinatura do método no repository aqui é o menos importante, afinal é complicado lembrar sem olhar a documentação oficial. Além do que, a solução poderia ser resolvida com JPQL ou SQL no repository.

* **Peso 7**: Consultar as transações a nível de banco de dados respeitando os filtros e ordenação;
* **Peso 2**: Preocupar-se em não carregar todas as transações do cliente para memoria de uma vez. Uso de paginação ou mesmo outra forma de limitar as linhas carregadas é aceitável (por exemplo, carregar somente as últimas 1k transações);
* **Peso 1**: Preocupar-se com as colunas carregadas na consulta. Nesse caso o uso de projeção em algum nível é o esperado;

## Resposta do Especialista:

* Como o repository já existe no projeto, eu não preciso criá-lo. Posso simplesmente adicionar um novo método na interface para fazer a consulta das transações no banco. A primeira versão desse método seria a mais simples possível, onde **eu buscaria todas as transações do cliente feitas nos ultimos 30 dias ordenadas por data de criação**, algo como:
* fun findByClienteIdAndCriadaEmGreaterThanEqualsOrderByCriadaEmDesc(  
   clienteId: Long,   
   criadaEm: LocalDateTime  
  ) : List<Transacao>
* A sacada agora é obter o clienteId do usuário logado na aplicação e calcular a data para 30 dias no passado (truncar a hora ajuda aqui). No final, para usar este método eu poderia fazer algo como:
* val clienteId = // obtem cliente do usuário logado por exemplo  
  val ultimos30Dias = LocalDateTime.now().minusDays(30).truncatedTo(DAYS)  
    
  val transacoes = repository.findByClienteIdAndCriadaEmGreaterThanEqualsOrderByCriadaEmDesc(clienteId, ultimos30Dias)
* Verifico no console da IDE se o SQL gerado pelo Micronaut foi gerado corretamente. Em caso de erros ou dúvidas, eu consulto a documentação oficial do Micronaut;
* Tendo certeza que a primeira versão da consulta funcionou, eu sigo com a próxima versão: **limitar o número de linhas carregadas do banco de dados**. O problema aqui é que eu não sei quantos transações um cliente pode ter em 30 dias, o que é muito perigoso pois eu poderia carregar milhares de linhas para memoria sobrecarregando o banco de dados, a aplicação e também o aplicativo mobile. Para resolver isso eu utilizo paginação, que me permite paginar o resultado da consulta de tal forma que carregaria somente um número limitado de linhas pequeno o suficiente para não prejudicar a aplicação, algo comum em frameworks de persistência e bancos de dados relacionais (SQL):
* fun findByClienteIdAndCriadaEmGreaterThanEqualsOrderByCriadaEmDesc(  
   clienteId: Long,   
   criadaEm: LocalDateTime,  
   pageable: Pageable // indica a pagina (index e size)  
  ) : Page<Transacao> // retorna as linhas daquela pagina
* Testo se a paginação funciona olhando o SQL gerado no console. A documentação oficial me ajuda aqui para acertar a sintaxe;
* Agora, para terceira e última versão do método, eu carrego somente os campos que aplicativo mobile precisa, que no caso são 3 campos apenas. A idéia aqui é **carregar somente os dados necessários do banco para otimizar o tempo de resposta e poupar recursos como rede, memoria e CPU entre aplicação e banco de dados**. Para isso, o Micronaut me permite trabalhar com projeções (projections), que nada mais são do que usar um DTO que possui somente os atributos que eu preciso em vez da entidade em si: ```kotlin @Introspected data class TransacaoParaMobile( val descricao: String, val valor: BigDecimal, val criadaEm: LocalDateTime )
* fun findByClienteIdAndCriadaEmGreaterThanEqualsOrderByCriadaEmDesc( clienteId: Long, criadaEm: LocalDateTime, pageable: Pageable // indica a pagina ) : Page // mudo o retorno para o DTO
* Por fim, testo a consulta mais uma vez para ter certeza que o SQL gerado e o resultado estão de acordo com o que espero;