## O que é o DataLoader?

O **DataLoader** é um componente usado em GraphQL para:

* Agrupar chamadas repetidas (batching)
* Evitar duplicação de consultas (caching por requisição)
* Resolver campos relacionais com eficiência

## O Problema N+1

Imagine que você está buscando uma lista de estudantes com suas matérias:

|  |
| --- |
| **graphql** |
| query {  students {  id  name  learningSubjects {  subjectName  }  }  } |

Internamente:

SELECT \* FROM students

Para cada student → SELECT \* FROM subjects WHERE student\_id = ?

Uma query GraphQL pode gerar múltiplas chamadas para dados relacionados

Se forem **100 estudantes**, teremos 1 query para os estudantes e **100 queries para as matérias.** **(1 + N) consultas.**

## Com DataLoader para learningSubjects

1. **Requisição GraphQL**

O cliente faz uma requisição como:

|  |
| --- |
| **graphql** |
| query {  getStudent(id: "1") {  id  address {  street  city  }  }  } |

1. **Interceptor registra os loaders**

|  |
| --- |
| **Interceptor** |
| @Component  public class StudentDataLoaderInterceptor implements WebGraphQlInterceptor {  public Mono<WebGraphQlResponse> intercept(...) {  DataLoaderRegistry registry = new DataLoaderRegistry();  registry.register("addressDataLoader", DataLoaderFactory.newMappedDataLoader(addressDataLoader));  ...  }  } |

1. **Resolver usa o DataLoader**

|  |
| --- |
| **Resolver** |
| @SchemaMapping(typeName = "StudentResponse", field = "address")  public CompletableFuture<AddressResponse> getAddress(StudentResponse student,  DataLoader<Long, AddressResponse> addressDataLoader) {  return addressDataLoader.load(student.getId());  } |

O que acontece:

* O método load(...) **não faz a query imediatamente, apenas agenda**
* O DataLoader **coleta todos os studentIds** requisitados
* Depois, executa uma query do tipo:

|  |
| --- |
| **SQL** |
| SELECT \* FROM address WHERE student\_id IN (1, 2, 3, ...) |

1. **BatchLoader executa o acesso real**

O BatchLoader é a **função principal de carregamento em lote** usada pelo DataLoader. Ele define **como buscar todos os dados de uma vez**, usando um Set<chave> (neste caso, studentIds) e retornando os dados agrupados por chave.

|  |
| --- |
| @Component  public class AddressDataLoader implements MappedBatchLoader<Long, AddressResponse> {  public CompletionStage<Map<Long, AddressResponse>> load(Set<Long> studentIds) {  // Poderia ser uma query real: SELECT \* FROM address WHERE student\_id IN (...)  Map<Long, AddressResponse> map = ... // simulate or fetch real  return CompletableFuture.completedFuture(map);  }  } |

1. **Resultado é mapeado e entregue**

Cada chamada a .load(id) recebe o resultado certo, pois o Map<Long, AddressResponse> é automaticamente particionado pelo DataLoader.

## Por que isso é poderoso?

* Reaproveita dados sem reexecutar queries
* Reduz drasticamente chamadas ao banco (reduz centenas de queries em uma so)
* Limpa os resolvers — nada de lógica de repositório neles
* Facilita testes unitários
* Tudo isso ocorre **dentro de uma única requisição**

A diagram of a data base

Description automatically generated