06

Para saber mais: DataLoader para Turmas

E aí, pensou no desafio de como implementar mais um DataLoader para as consultas feitas à tabela `turmas`? Para este desafio em especial, vamos ver uma possível solução.

Para resolvermos o objeto `Matricula` que era retornado no campo `matriculas` do tipo `User`, implementamos a resolução no resolver de `User` - o que fizemos no vídeo.

Mas as chamadas para a tabela `turmas` estão sendo feitas a partir de um *subcampo* de `Matricula`. Vamos rever a query feita no playground:

```
query {
  user(id: 10) {
    nome
    matriculas {
```

```
        turma {
          descricao
        }
      }
    }
  }
```

COPIAR CÓDIGO

Pensando que o DataLoader deve ser implementado na chamada do resolver — afinal de contas, **é justamente o que queremos impedir, que o resolver seja chamado toda vez para cada consulta**—, o subcampo `turma` na query acima está sendo resolvido dentro de ‘matriculaResolvers.js`. Então vamos lá.

Em `./api/matricula/resolvers/matriculaResolvers.js`, está sendo usado o método `getTurma()` para resolver o campo. Porém não vai ser possível implementar o DataLoader diretamente neste método, pois ao contrário do que foi feito no método `getMatriculasPorEstudante()`, que é chamado somente na resolução do campo `turma`

dentro do tipo `Matricula`, aqui o método `getTurma()` é utilizado em outras partes do código — por exemplo, para fazer a query `turmas` que foi definida no schema de turmas.

Então vamos criar mais um método à parte para utilizar o DataLoader, e chamar de `getTurmasCarregadas()`. Em `./api/matricula/resolvers/matriculaResolvers.js`, teremos:

```
  Matricula: {
    // . . . outros resolvers,
    turma: (parent, _, { dataSources })
dataSources.turmasAPI.getTurmasCarregad
  }
```

COPIAR CÓDIGO

Agora, criamos esse novo método em `./api/turma/datasource/turma.js`:

```
const DataLoader = require('dataloader'
```

```
  // . . . restante dos métodos


  getTurmasCarregadas = new DataLoader(a
    const turmas = await this.db
      .select('*')
      .from('turmas')
      .whereIn('id', idsTurmas)


    return idsTurmas
      .map(id => turmas
        .find(turma => turma.id === id))
  })
```

COPIAR CÓDIGO

Um detalhe importante sobre o método acima: vamos lembrar que `getTurmasCarregadas()` vai ser chamado pelo resolver cada vez que o nível de `turma` na resolução da query precisa ser resolvido! O fluxo de resolução "nível por nível" do GraphQL continua sendo executado. Então, ao invés de `.filter()`, que sempre retorna os

resultados em uma array, agora utilizamos o `.find()`, que retorna somente o objeto.

Agora, o log de consultas ao banco mostra apenas uma consulta na tabela `turmas`:

```
knex:query select * from `turmas` where
```

COPIAR CÓDIGO

Você pode testar outras alternativas em seu projeto!