02

Quem chegará primeiro?

Transcrição

Já sabemos que em algum momento vamos precisar editar as informações sobre os produtos na nossa loja, muito provavelmente em uma promoção. Imagine a seguinte situação: Nossa loja precisa publicar promoções diariamente e, também, realizar edição dos produtos. No caso de uma loja que possui vários funcionários, o que irá acontecer caso dois funconários **atualizem as informações sobre algum produto ao mesmo tempo**? Qual dos dados irá valer? O que for salvo primeiro ou por último? Como a nossa aplicação deverá lidar com isso?

Para ilustrar melhor o problema imagine que nossa loja irá lançar uma promoção para os produtos de tecnologia, abaixando seus preços em 10%. Um funcionário chamado Fábio vai até o site e realiza a alteração do preço do livro **Arquitetura e Design de Software: Uma abordagem sobre a plataforma Java**. Enquanto isso, Rômulo, outro funcionário, resolve mexer no título do mesmo livro, deixando-o mais simples. Portanto, ele irá alterar o nome do livro para **Arquitetura e Design de Software**.

Pouco tempo depois de Fábio abrir a página de edição e começar a alteração do preço do livro, Rômulo abriu a mesma página para editar o mesmo livro e alterar seu nome. Supondo que Fábio confirme a edição primeiro, quando Rômulo tentar editar o mesmo livro o que irá acontecer com o novo preço de Fábio? Será **perdido**! Isso é o que chamamos de *[Conflito escrita-escrita][1]*.

O que é Lock pessimista?

Podemos lidar com esse cenário de algumas formas: Uma abordagem pessimista seria impedir que aconteça o conflito e pedir para o banco de dados travar o registro que está sendo atualizado.

O JPA dá suporte ao **Lock Pessimista**. O `EntityManager` possui um método `lock` que trava o registro no banco de dados. Assim apenas uma pessoa pode acessar esse registro ao mesmo tempo. Um exemplo de como implementar o **Lock Pessimista** segue abaixo:

```
EntityManager em1 = // obtem EntityManager
EntityManager em2 = // obtem EntityManager

em1.getTransaction().begin();
em2.getTransaction().begin();

Produto produtoDoEM1 = em1.find(Produto.class, 1);
em1.lock(produtoDoEM1,LockModeType.PESSIMISTIC_WRITE);

produtoDoEM1.setNome("Maria");

Produto produtoDoEM2 = em2.find(Produto.class, 1);
em2.lock(produtoDoEM2, LockModeType.PESSIMISTIC_WRITE); // aplicação trava aqui
```

Logo após buscar `Produto` do banco, pedimos uma trava do registro no banco através do método `lock` . Abaixo disso, tentamos buscar o mesmo usuário usando um outro `EntityManager` e pedimos sua trava. Nesse momento, a `Thread` corrente travará pois há alguém que já está acessando o registro (e travou o registro).

É importante mencionar que o banco precisa suportar esse tipo de lock. O `EntityManager` apenas passa o comando para o banco e é ele quem realmente cuidará de travar o segundo `EntityManager`.

Problemas com Lock Pessimista

Essa solução tem como consequência um grande gargalo de escalabilidade, já que no momento em que travamos um registro os demais ficarão esperando a liberação do mesmo para realizar a edição. Além disso, o banco de dados deve oferecer suporte ao *lock* (como já foi mencionado). Esse recurso deve ser utilizado com muito cautela dentro de uma aplicação web e normalmente outras soluções são utilizadas.