Transcrição

Ainda na parte de transição de estados, quando instanciamos, persistimos a entidade e ela fica no estado de managed, nós encontramos uma situação no último vídeo que é a atualização. Nós não tínhamos visto ainda como atualizar uma entidade. Só havíamos colocado, nas classes DAO, um método para cadastrar uma categoria ou um produto, `public void cadastrar(Categoria categoria) {` , mas não vimos como faz para atualizar.

Quando fizemos a simulação, já era possível ver uma atualização. Vamos rodar novamente ("Run As > 1 Java Application). Analisando o Console, perceberemos que ele faz o *insert* e, na sequência, um *update*, isto é, já está atualizando uma entidade.

Toda vez que a entidade está no estado managed, está gerenciada, qualquer mudança que fizermos em algum atributo,a JPA detectará e, no `commit()` ou no `flush()` do `EntityManager` , ela vai, automaticamente, sincronizar essas mudanças no banco de dados, porque sabe que é necessário fazer o *update* no banco de dados.

Portanto, é assim que funciona o *update* no banco de dados, basta pegar uma entidade que esteja no estado managed e alterar os atributos dessa entidade. Quando fizermos o `commit()` da transação ou um `flush()` manualmente, esse estado será sincronizado automaticamente com o banco de dados.

Mas, o problema é que não sabemos se a entidade está no estado managed, talvez ela já esteja no estado detached se chamarmos o `clear()` ou se fecharmos o `EntityManager` com o `close()` . Nesta situação, o *uptade* não acontecerá.

Em `CadastroDeProduto.java`, nós havíamos alterado o nome, commitamos, fizemos o `close()` do `EntityManager`, e alteramos o nome da entidade - da categoria - para `"1234"`, e um segundo *update* não foi disparado.

```
        EntityManager em = JPAUtil.getEntityManager();
        em.getTransaction().begin();

        em.persist(celulares);
        celulares.setNome("XPTO");

        em.getTransaction().commit();
        em.close();

        celulares.setNome("1234")
    }

}
```

COPIAR CÓDIGO

Depois de fechado o `EntityManager`, ele está no estado detached, e, neste estado, nada que alterarmos na entidade será sincronizado automaticamente com o banco de dados. Então, surge a questão de como voltar a entidade para o estado managed.

Se ao invés de fecharmos o `EntityManager`, escrevermos `clear()`, o `EntityManager` ainda estará aberto, quer dizer que ainda podemos trabalhar com ele, porém, com o `clear()` nós tiramos todas as entidades, todas estão detached. Como fazemos para voltar a entidade para o estado managed? Pois, se quisermos atualizar uma informação em `celulares.setNome("1234");` ela não será atualizada.

Ainda considerando o nosso exemplo anterior, fizemos um `clear()`, alteramos o nome e, agora, vamos dar um `flush()`, isto é, `em.flush();`. Vamos também tirar o `commit()` e trocar por `flush()`, porque ainda não queremos commitar a transação, mas, queremos sincronizar com o banco de dados. Será que agora ele fará dois *updates* ou apenas um? Vamos rodar ("Run As > 1 Java Application").

```
        EntityManager em = JPAUtil.getEntityManager();
        em.getTransaction().begin();

        em.persist(celulares);
        celulares.setNome("XPTO");

        em.flush();
        em.clear();

        celulares.setNome("1234")
        em.flush();
    }

}
```

COPIAR CÓDIGO

No Console, notaremos que ele fez apenas um *update*, que foi `celulares.setNome("XPTO")`, quando mudamos o nome para `"XPTO"`. No `flush()`, ele disparou um *insert* e um *update* (que havíamos persistido e mudado o nome), mas demos um `clear()` e, agora, a entidade não está mais gerenciada. Então, por mais que tenhamos alterado um atributo, quando chamarmos o `flush()`, ele não vai sincronizar.

Então, o que precisamos fazer se quisermos voltar para o estado managed? Existe outro método que não havíamos estudado ainda, o `merge()`, que tem como objetivo pegar uma entidade que está no estado detached e retorná-la ao estado managed

(gerenciado).

A partir dali, qualquer mudança que fizermos na entidade será analisada e sincronizada ao banco de dados quando realizarmos o `commit()` da transação ou `flush()` .Vamos simular essa situação. Nós fizemos o `clear()` e a entidade está no estado detached.

Agora, vamos chamar `em.merge()` , passando a entidade `celulares` , que, então, volta para o estado managed. Continuando, alteraremos o nome, e faremos um `flush()` . Portanto, ela deveria fazer dois *updates*. Vamos rodar e verificar se isso de fato acontecerá.

```
        EntityManager em = JPAUtil.getEntityManager();
        em.getTransaction().begin();

        em.persist(celulares);
        celulares.setNome("XPTO");

        em.flush();
        em.clear();

        em.merge(celulares);
        celulares.setNome("1234")
        em.flush();
    }

}
```

COPIAR CÓDIGO

Ao observar o Console, perceberemos que ele nos mandou uma *exception*, "javax.persistence.PersistenceException" e indicou algo importante: "No default constructor for entity: : br.com.alura.loja.modelo.Categoria". Significa que a entidade `Categoria` não tem um construtor *default*.

Retornando à entidade `Categoria.java`, tínhamos criado o seguinte construtor:

```
public Categoria(String nome) {
      this.nome = nome;
  }
```

COPIAR CÓDIGO

Com a intenção de, na hora de dar `new` na categoria, passar também o nome. Mas a JPA precisa que as entidades tenham um construtor padrão. Até então, ela não havia reclamado disso, porque estávamos fazendo apenas *insert*, mas quando chamamos um `merge()`, ele faz um *select* no banco de dados. Ao carregar a entidade do banco de dados e criar o objeto, a JPA precisa do construtor *default*.

Logo, precisamos inserir o construtor *default* nas entidades, tanto na `Categoria.java` quanto na `Produto.java`. Assim, na `Categoria.java` teremos:

```
public Categoria() {
    // TODO Auto-generated constructor stub
}
```

COPIAR CÓDIGO

E no `Produto.java`:

```
@ManyToOne
private Categoria categoria;

public Produto() {
    // TODO Auto-generated constructor stub
}
```

COPIAR CÓDIGO

Vamos rodar mais uma vez nossa classe `CadastroDeproduto.java` e analisar se ele fará dois *updates* agora. No Console, reopararemos que ele fez um *insert*, o primeiro *update* (do nome "XPTO") e fez o *select*, por causa do `merge()`, porém, não fez o *update* referente à mudança de nome para `"1234"`.

Isso aconteceu, porque quando chamamos o método `merge()` e passamos uma entidade, ele não muda o estado dessa entidade para managed, ele devolve uma nova referência, e esta sim, estará no estado managed. Mas, a que passamos como parâmetro, no nosso caso, `"celulares"`, continua detached.

```
em.merge(celulares);
celulares.setNome("1234")
em.flush();
```

COPIAR CÓDIGO

Por isso, quando mudamos o atributo, não adiantou nada, já que fizemos essa mudança na entidade que ainda está detached. Sendo assim, se desejarmos mudar o atributo, é necessário criar uma nova categoria e atribuir, ou, para mudar de fato o objeto, precisamos fazer `celulares = em.merge(celulares);` ("celulares", que é o nosso objeto, agora aponta para o retorno do método merge()). Ou seja, o método `merge()` devolve a entidade no estado managed.

Vamos rodar e analisar o Console. Agora ele fez o *insert*, o *update*, o *select* do `mege()` e o *update* da atualização, já que, agora, sim, estamos trabalhando em cima da entidade que está managed. É assim que o método `merge()` funciona.

Comumente, nos projetos, temos um método para atualizar. Funciona assim: temos um método para cadastrar e estamos na categoria DAO e teremos um método para atualizar uma entidade.

```
    public void cadastrar(Categoria categoria) {
        this.em.persist(categoria);
    }

    public void atualizar(Categoria categoria) {
        }

}
```

COPIAR CÓDIGO

O que esse método faz? Em teoria, ele não precisa fazer nada, pois já recebe a entidade com as informações alteradas, mas, como não sabemos se essa entidade está managed, nós, de certa forma, a forçamos a ficar managed. Então, colocamos `this.em.merge(categoria)`, só para garantir que essa categoria estará no estado managed.

```
public void atualizar(Categoria categoria) {
    this.em.merge(categoria);
}
```

COPIAR CÓDIGO

Não há necessidade de alterar os atributos, porque eles já chegam atualizados, isto é, já chegou uma categoria detached com todos os atributos atualizados, então, quando chamarmos o `merge()`, ele apenas a coloca no estado managed e, depois, quando fizermos o `flush()` da transação, ele disparará o *update* automaticamente.

Aparentemente, não precisamos do método `merge()`, porque sua função não é atualizar, mas, sim, para o caso de, se por um acaso a entidade estiver detached, o método `merge()` a voltará para o estado managed. Para atualizar no banco de

dados, vamos: carregar a entidade do banco, mudar o atributo, commitar a transação. E, pronto, já está managed.

Quando carregamos do banco de dados, ela já está managed. Então, se alterarmos qualquer atributo e fizermos o `flush()` ou o `commit()`, ele fará a sincronização com o banco de dados (fará o *updtae* automaticamente).

No próximo vídeo, falaremos de outros estados das entidades da JPA. Vejo vocês lá!!