02

# Usando PreparedStatement

## Transcrição

[00:00] Fala, aluno. Tudo bom? Voltando ao nosso curso de JDBC, eu gostaria de passar novamente na nossa classe `TestaInsercao`. Nessa classe, nós já realizamos alguns testes e vimos que todos foram feitos com sucesso. Porém, agora, eu queria desenvolvê-la de forma um pouco diferente.

[00:21] Vamos criar uma nova classe, chamada "TestaInsercaoComParametro", e vocês já vão entender qual a motivação dessa classe. Dentro dela nós vamos ter um método `public static void main(String[] args)` e para não perder muito tempo, vamos copiar esse mesmo código da classe `TestaInsercao` na `TestaInsercaoComParametro`.

[00:46] Vamos adicionar o `throws SQLException` na `TestaInsercaoComParametro`. Minha motivação para desenvolver essa classe é pensando o seguinte: se o sistema, um dia ele tiver uma interface gráfica e as informações como o nome e a descrição do produto forem informados através de um formulário, por exemplo, o que vai acontecer?

[01:11] Quando o usuário colocar no meu formulário o nome e a descrição e dar um submit, do meu lado da aplicação, eu tenho que recuperar essa informação, guardar os valores em uma variável e concatenar com a cláusula SQL, que aí, de fato, quando for executado o comando `stm.execute`, ele vai inserir as informações com as informações que usuário nos passou.

[01:38] Então eu vou pegar, em `TestaInsercaoComParametro`, uma variável chamada `String nome "";`, do tipo string, por enquanto eu vou iniciar essa

variável com uma string vazia, descrição também do mesmo jeito: `String descricao = "";` . Agora eu só preciso concatenar essas strings na nossa cláusula insert do SQL.

[02:05] Então fica `stm.execute("INSERT INTO PRODUTO (nome, descricao) VALUES (' " + nome + " ', ' " + descricao + " '), Statement.RETURN_GENERATED_KEYS");` . Se eu inicializar com `String nome = "Mouse";` no nome e `String descricao = "Mouse sem fio";` na descrição, não é para ter nenhuma diferença da forma que estava antes.

[02:28] Nós vimos, ele criou um novo ID e na nossa classe `TestaListagem` , o produto foi criado da mesma forma. Só que agora nós estamos lidando com informações que vem de um formulário, que vem de um usuário. O usuário, ele pode ter, por exemplo, errado na hora de inserir o nome, e passou `String nome = "Mouse' ";` , com aspas simples.

[03:00] Novamente, o nome do produto está entre aspas duplas na nossa aplicação, não é para ter nenhum tipo de problema. Se eu mandar executar o `TestaInsercaoComParametro` , eu quero o ID 41. Tivemos um erro. Que erro é esse? Você tem um erro na sua sintaxe SQL. Mas o mais estranho disso é que nós não mudamos nada na nossa string.

[03:27] Na verdade o usuário errou, passou uma aspa simples junto com o nome do produto, mas era pra ter sido tratado como uma string para o SQL, era para ter salvo uma string "Mouse' ", com aspas simples. Estranho. Vamos verificar o que aconteceu. Eu vou criar uma variável chamada `sql` , em "TestaInsercaoComParametro", vai ser do tipo string também, e eu vou guardar a nossa cláusula insert do SQL nela.

[04:00] Então vai ser aqui `string sql = "INSERT INTO PRODUTO (nome, descricao) VALUES (' " + nome + " ', ' " + descricao + " ');` . Agora, invés de passar para o `stm.execute` a string, eu passo `stm.execute(sql);` , com a variável. Só que para verificar como a aplicação está montando a nossa

cláusula SQL, eu vou dar um `System.out.println(sql);` na variável. O erro vai ser o mesmo, só que pelo menos agora conseguimos verificar como está a cláusula.

[04:30] Ele montou, no `VALUES`, e quando ele foi montar a cláusula, na verdade ele considerou a aspas simples, colocada por engano no nome, em vez de ser parte da string e salvar essas aspas simples como string, as aspas simples, ela é específica do SQL. Então, nos comandos SQL, quando nós vamos inserir uma string, a string ela fica, diferentemente do Java, que é entre aspas duplas, a string vai ficar entre aspas simples no SQL.

[05:08] Foi por isso que a nossa aplicação deu erro. Como as aspas simples são do domínio do SQL, dos comandos SQL, da linguagem SQL, então o erro do usuário acabou quebrando a nossa aplicação. Só que a pior parte não é o erro. Vamos supor, aqui foi um erro de usuário, o usuário é humano, suscetível ao erro.

[05:32] Mas nessa situação, nós encontramos também usuários que sabem o que estão fazendo e fazem para prejudicar o desenvolvedor, o dono do sistema, porque ele pode fazer isso, quando ele terminar de fazer a descrição que ele quer para o produto, ele pode fechar parênteses, dar um ponto e vírgula e, por exemplo, escrever uma cláusula delete - "Mouse sem fio); delete from Produto;".

[06:07] Se ele conhecer um pouco da arquitetura, se ele conhecer um pouco das tabelas, souber que tem uma tabela produto, se ele der um "delete from Produto", na hora que ele der um submmit, em vez de ele criar um novo produto, ele vai deletar toda a nossa tabela de produto. Olha só que prejuízo.

[06:28] Por causa de um simples comando que o desenvolvedor muitas vezes não está esperando, porque o usuário, ele está inserindo uma cláusula que está realizando uma ação que não é esperada. Eu espero que, nesse meu formulário, eu insira produtos e não delete os meus produtos.

[06:53] Então com o usuário um pouco mais experiente, um hacker, ou uma pessoa que está agindo de má-fé, ela consegue injetar um SQL, uma cláusula SQL e consegue quebrar a sua aplicação, te dar um prejuízo bem grande, principalmente se tratar-se de um delete. Essa ação, ela se chama exatamente como eu falei ainda agora, é uma injeção de SQL, é uma SQL Injection.

[07:26] Isso é uma prática até comum, as pessoas, elas ficam procurando vulnerabilidades nas aplicações e ficam fazendo testes para verificar, muitas vezes com a intenção de recuperar alguma informação valiosa, ou muitas vezes apenas para te dar o prejuízo mesmo, como no caso de um delete. A pessoa pode não recuperar nenhuma informação valiosa, mas também ela apaga a sua informação valiosa. Então como eu posso tratar esse problema?

[07:56] Sendo que com um pequeno erro no nome, que é uma aspa simples, eu vou quebrar a minha aplicação, e um pouco mais grave, um pouco mais grave não, bem mais grave eu deleto uma tabela toda. O que posso fazer no código, eu como desenvolvedor, é fazer uma validação caractere por caractere. Então, quando eu verificar que tem uma aspa simples, eu escapo, tudo o que for à direita, depois das aspas, eu não considero.

[08:27] Considero também que tudo o que tiver um ponto e vírgula, eu não vou considerar e não vou contar o que estiver à direita, após o ponto e vírgula. Mas olha o tanto que parece ser falho isso, porque eu sou humano. Eu, desenvolvedor, sou humano, então estou suscetível a erro também. Assim como o usuário está, ao inserir, ele está suscetível a erro e quebrar a minha aplicação sem querer, eu posso quebrar fazendo uma validação manual.

[08:53] Porque eu posso inserir, eu posso criar uma nova classe de inserção e esquecer de fazer essa validação, eu posso, muitas vezes, fazer essa validação errada. Então o ideal é que não façamos isso. Então, como eu faço? Graças ao JDBC, nós temos uma maneira de criar - na verdade não criar, validar essas informações de forma que o JDBC me provê a solução.

[09:19] O tempo todo nós vínhamos criando Statements. Uma vez que criávamos um Statement, nós passávamos a Query concatenando valores ou , por exemplo no nosso caso do Insert, nós passávamos antes já o produto e a descrição direto, dentro do Values, como uma string. Só que agora, eu vou mudar esse comportamento.

[09:45] Invés de eu criar o Statement, eu vou preparar um Statement. Quando eu preparo um Statement, eu estou falando que a responsabilidade de gerenciar esses atributos que eu passo para a minha cláusula SQL, não vai ser mais eu que vou fazer, agora vai ser o JDBC. Então eu apago todas essas linhas anteriores, que eu tinha feito para exemplificar, do `Statement sql` no `TestaInsercaoComParametro .`

[10:16] Aqui, em `VALUES` , eu não vou ter mais nada disso, eu só vou falar o seguinte: dentro dessa minha cláusula SQL, esse meu Insert, ele vai receber dois parâmetros. Esses parâmetros serão o nome e a descrição. Como que eu vou saber, como que o `.prepareStatement` vai saber que essa interrogação e essa interrogação, em `VALUES (?, ?)` , vão ser substituídas pelo nome e a descrição?

[10:41] Quando estamos usando o `.prepareStatement` , ele vai nos retornar um `PreparedStatement` , e aqui começa a beleza da coisa. Quando eu quero *setar* esse atributo `String nome` , nessa primeira interrogação do `VALUE` , eu só preciso falar para ele o seguinte: `PreparedStatement` , *seta* uma string, no primeiro atributo, que vai ser o valor da variável `nome - stm.setString(1, nome); .`

[11:15] Fica bem legível, você bate o olho e você já consegue entender. Eu faço a mesma coisa para o segundo parâmetro de `VALUE` , para a segunda interrogação, só que agora eu falo `stm.setString(2, descricao); .` A maravilha disso é que quando eu preparo o meu Statement, eu estou falando para ele substituir essas interrogações de `VALUE` , pelo `nome` e `descrição` , eu não preciso mais me preocupar em passar o `stm` para o `execute` .

[11:47] Eu só preciso executar, a Query, ela já está preparada. Eu só preciso agora tirar esse `Statement.RETURN_GENERATED_KEYS` de dentro do `execute` também. Ele vai ficar `stm.execute();` , sem nenhum parâmetro e eu dou uma vírgula no final de `connection.prepareStatement("INSERT INTO PRODUTO (nome, descricao) VALUES (?, ?)", Statement.RETURN_GENERATED_KEYS);` .

[12:06] E falo que eu vou querer também que as chaves geradas sejam retornadas após a inserção. O código `TestaInsercaoComParametro` todo refatorado, compilando e agora nós só precisamos executar. Agora sim eu tenho que ter o meu ID gerado, de número 41. Vimos que o ID foi criado.

[12:32] Se eu listar os produtos, em `TestaListagem` , nós vamos ver que foi criado o produto da forma que nós tínhamos deixado, então agora, sem a nossa validação manual, sem precisarmos fazer intervenção naquele nome com aspas simples, o próprio `PreparedStatement` , ele já se preocupou e já identificou que é uma string, a aspa simples ela é uma string, foi um erro do usuário, ele não quer quebrar a aplicação, então ele vai tratar como string.

[13:06] A mesma coisa na descrição, ele não tratou o delete como um delete, ele tratou o delete como uma string. Então aqui está a maravilha do `PreparedStatement` . Além de ele evitar os SQL Injections, ele deixa o código bem mais legível, então vemos agora, se você bater o olho em `VALUES` , eu tenho dois atributos e embaixo eu *seto* esses atributos. Então ficou muito mais bonito, fica muito menos vulnerável o nosso código.

[13:37] Além de mais bonito, menos vulnerável, então é muito vantajoso. Com isso, agora nós estamos aptos a fazer o Refactory de todas as classes que nós criamos até agora. Então, para isso, a `TestaListagem` , a `TestaRemocao` , invés de usarmos o Statement, agora nós podemos passar a utilizar o `PreparedStatement` . Então é isso, aluno. Espero que tenham gostado e até a próxima aula.