05

Configurando o banco

Transcrição

[00:00] Então vamos voltar ao navegador. Entrando no Maven Repository, temos o MariaDB Java Client que é o *driver* do MariaDB para o Java. E aqui eu tenho todas as versões de *drive* disponível até o momento. Vou selecionar a última versão de *driver* disponível, vou copiar o comando a dependência do Maven, vou voltar a minha IDE e vou colar aqui. Após salvar, o Maven já vai fazer o download deste *driver* para dentro das nossas dependências.

[00:42] Agora eu já tenho a dependência para comunicar com o banco de dados e eu já tenho a dependência do *driver* do banco de dados que realmente eu quero conectar. Mas agora eu preciso informar para o Spring qual é a URL, o usuário e a senha as configurações para realizar essa conexão com esse banco de dados. Aí sim vem o arquivo *properties*. Para que temos esse arquivo?

[01:11] Aqui colocamos propriedades e configurações para utilizar no Spring. Eu já criei um arquivo de configuração para conectar no MariaDB. Vou copiar e colar e já vou passar para vocês linha a linha o que cada um significa para que possamos ter mais agilidade. Esse é um arquivo de configuração para conectar no MariaDB.

[01:43] Na primeira linha nós informamos para o DataSource onde está o banco de dados, onde ele está comunicando. Então ele é JDBC e na minha instalação eu o coloquei na "127.0.0.1" que é o LocalHost na porta "3306", instalação *default*. E no final eu coloco uma barra e o nome do banco de dados que eu quero. Aqui eu coloquei `alura`, mas você pode colocar o nome do banco de dados que você preferir.

[02:16] Aqui é só o nome de dados da sua preferência. Feito isso nós temos o *username* que é para passar qual o usuário que você criou na instalação e o *password* para você adicionar a senha que você colocou na instalação. Logo em seguida nós temos o TestWhileIdle e o ValidationQuery. Essas duas linhas são para fazer testes da configuração de conectividade do usuário, senha e da URL que você passou.

[02:50] Em seguida nós temos que falar qual é o nome do *driver* que vamos estar utilizando e com isso nós finalizamos a configuração de DataSource. Agora nós vamos a configuração de JPA. Na configuração de JPA, esse `show-sql=false`, eu deixei ele como falso por quê? O que ele faz?

[03:10] Cada ação que o Framework toma com o banco de dados, o JPA, por exemplo, um *insert* ou um *delete,* ele gera comandos SQL. Se você deixar como *true*, ele vai printar no console cada comando que ele está executando com o banco de dados. Se caso você queira visualizar o que ele está fazendo. Aqui para mim eu não quero visualizar isso no momento, então eu deixei como *false*. O `ddl-auto=update` é por quê?

[03:39] Nós vamos criar entidades que o Framework vai transformar em tabelas. Então, por exemplo, se você tem uma entidade e quer adicionar um novo atributo a essa entidade, com o `ddl-auto=update`, ele vai pegar essa tabela que já existe e vai adicionar um novo campo sem destruir a tabela e sem destruir os registros que estão na tabela assim.

[04:03] Se você colocar, por exemplo, *create* ele vai pegar a tabela que já existe, vai deletar ela com todos os registros que já existem e vai criar uma tabela do zero com o novo atributo. Aqui nós temos também o `naming-strategy`. Por que nós utilizamos o `naming-strategy`? Quando nós programamos em Java, utilizamos o CamelCase o padrão de estrutura de escrever código em Java.

[04:32] Então, por exemplo, se eu for criar uma variável chamada `aluraDatabase`, em Java ele ficaria com algo parecido com isso que está na tela. O `naming-strategy` vai fazer o quê? Ele vai converter isso que está em um

padrão mais Java para algo padrão mais SQL. Então ele transformaria a nossa variável para algo mais assim. E por fim nós temos também o `dialect`.

[05:04] Às vezes nós temos alguns comandos SQL que funcionam em um determinado DataBase e não funcionam em outro. Por exemplo, um sistema de comando que funciona no SQL Server, mas não funciona no MySQL. Então por isso você tem que informar para o JPA qual é o dialeto do banco de dados que você está utilizando. Feito isso nós já finalizamos a configuração da nossa aplicação e eu vou entrar no DBeaver.

[05:35] Eu já tinha uma base de dados, vou excluir ela. Vamos fazer do zero para eu mostrar para vocês como que faz para conectar no DBeaver. Agora que estou zerado, aqui no Plug, no canto superior esquerdo da tela na aba "Navegador de bancos" você tem um ícone chamado "Nova Conexão de Bancos". Clique nela.

[05:58] Ja nova janela que se abre, você vai adicionar a opção “MariaDB”, clicar em "Seguinte" e na próxima janela você tem na aba “Main” o ServerHost que no meu caso é "localhost", a porta "3306", o usuário e a senha, que no meu caso é "root" também. Clicando em “Test Connection...", o meu já está conectando. Depois vou clicar "Terminar". Na janela lateral da aba "Navegador de bancos" le vai dar essa opção do LocalHost. Você clicando na seta, ele abre todas essas opções: banco de dados, usuário, administrador, informações do sistema.

[06:30] Aqui em banco de dados, você clica com o botão direito do seu mouse e vai na opção “Create New Database" e na janela você tem que colocar o mesmo nome de Database que você já criou lá na sua configuração, que no meu caso é `alura`. Então eu dou um "Ok" e ele cria o Database na lista lateral.

[06:51] Agora vamos voltar na nossa Package Java dentro da Package principal, na nossa classe “SpringDataApplication”. Selecione ela com o botão direito, vá em “Run As” e selecione a opção “Java Application”. Clicando nela a sua aplicação vai começar a *startar*.

[07:19] Então aqui a nossa aplicação já começou a *startar*. Cadê o console? O console está aqui. Vou maximizar para que possamos ver. Então ele está iniciando, iniciando, ele iniciou o Spring Data e inicializou o JPA. Então já temos aqui uma aplicação que está conseguindo conectar com o JPA. Muito obrigado por verem esse vídeo. Nos vemos na próxima.