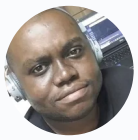
Artigos > **Programação**

Conhecendo o JDBC



André Bessa
Atualizado em 06/08/2021

Esse artigo faz parte da
Formação Java e Orientação a Objetos

Para começar a falar sobre JDBC, vamos pensar na seguinte situação: você é contratado para desenvolver uma aplicação em Java do tipo desktop. Como desafio, você precisa fazer a

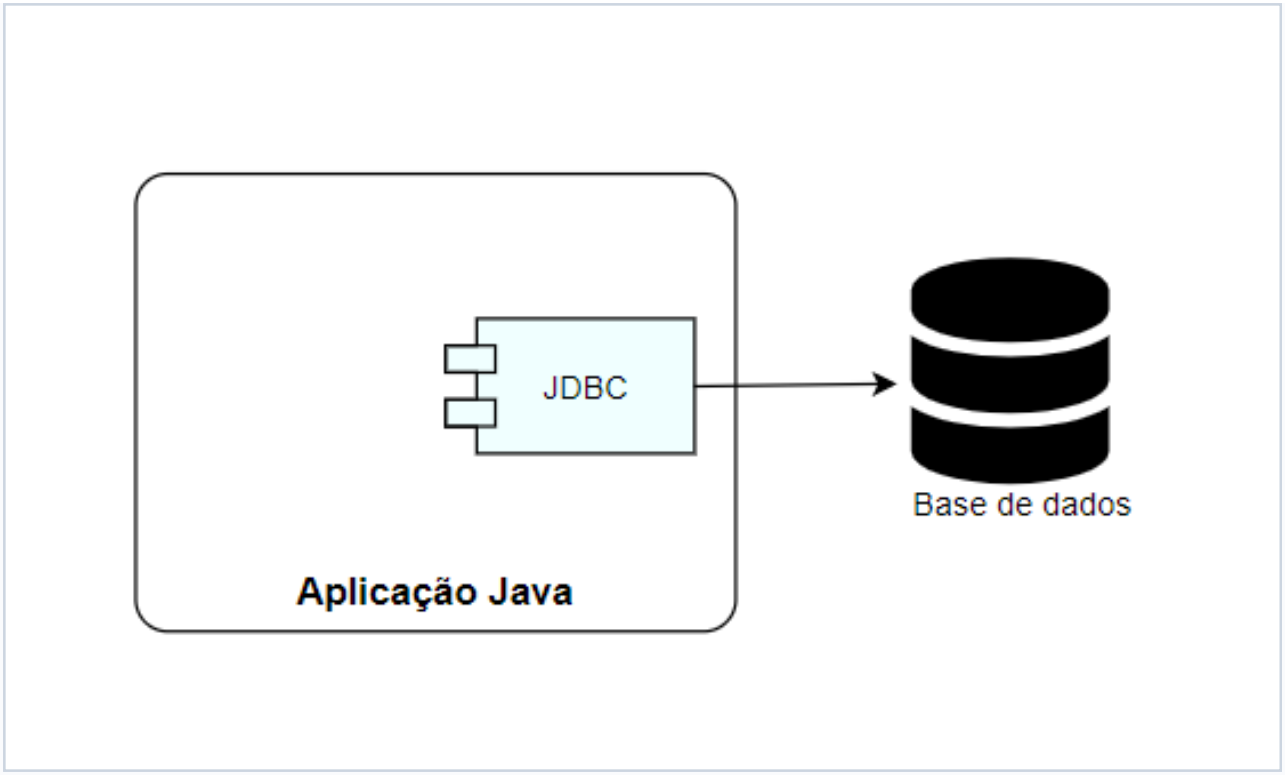
integração da aplicação com um banco de dados MySQL, que executa localmente, e outro banco SQL Server, rodando em um servidor Windows.

Como fazer isso? Se for usar linguagem Java, uma opção será a utilização dos recursos da API JDBC.

Os bancos de dados relacionais ainda representam a grande maioria das soluções de dados encontradas no mercado. Criar um sistema é uma atividade que, em algum momento, vai exigir que você acesse uma base de dados desse tipo. Por isso, diversas linguagens de programação implementam mecanismo de acesso a dados.

O que é JDBC?

A JDBC é uma API do Java que possibilita que uma aplicação construída na linguagem consiga acessar um banco de dados configurado local ou remotamente. A API é composta pelos pacotes `java.sql` e `javax.sql`, incluídos no JavaSE. Por meio das classes e interfaces fornecidas por esses dois pacotes, as pessoas podem desenvolver softwares que acessem qualquer fonte de dados, desde bancos relacionais até planilhas.



Componentes

A API JDBC é composta por dois componentes centrais. Em primeiro lugar, podemos falar dos pacotes (*Java.sql e Javax.sql*) que contêm as classes e interfaces que padronizam a comunicação da aplicação Java com uma base de dados

Outro item importante são os drivers, verdadeiros responsáveis pela conexão e interação com um banco específico. Um driver JDBC é uma classe que implementa a interface *java.sql.Driver*. Muitos drivers são totalmente desenvolvidos com o uso de Java, o que colabora para serem carregados de maneira dinâmica.

Os drivers também podem ser escritos de forma nativa, acessando outras bibliotecas ou outros drivers de sistema que permitam acesso a uma base de dados determinada.

A classe *DriverManager* define um conjunto básico de operações para a manipulação do driver adequado para a conexão com um

banco. Além disso, ela também é responsável por realizar a conexão inicial.

Tipos de drivers

Todo driver JDBC deve dar suporte mínimo às funcionalidades especificadas no padrão ANSI2 SQL-92. Através do driver, a aplicação Java acessa as implementações de classes e interfaces que vão permitir a execução dos comandos SQL em uma base de dados.

Hoje a arquitetura do JDBC possui quatro tipos de drivers diferentes:

- **Tipo 1:** A JDBC-ODBC possibilita o acesso a drivers do tipo ODBC, um padrão já consolidado para o acesso a bases de dados.
- **Tipo 2:** Neste tipo de driver é implementado o protocolo do proprietário do banco de dados. Ele transforma as chamadas JDBC em chamadas do banco com o uso da API proprietária.
- **Tipo 3:** Este tipo de driver faz a conversão das chamadas JDBC em outras chamadas do banco de dados, que são direcionadas para uma camada intermediária de software, ou *middleware*. Assim, a chamada será convertida para o protocolo do banco.
- **Tipo 4:** São escritos puramente em Java e implementam o protocolo proprietário do banco de dados. No geral, têm desempenho superior, já que acessam diretamente o *SGBD*, sistema gerenciador de banco de dados.

Usando o JDBC

Agora vamos ver de maneira prática como utilizar o JDBC para implementar uma conexão com uma base de dados. Nos exemplos abaixo, utilizamos o postgresql como fonte, mas a ideia central pode ser replicada para outros bancos de dados relacionais.

Como numa receita de bolo, vamos seguir um roteiro básico para a manipulação dos dados. Nossos passos iniciais serão:

- Definir a classe que implementa o driver JDBC, vamos usar *org.postgresql.Driver*;
- Definir a *string de conexão* do banco de dados, basicamente uma cadeia de caracteres com informações para conectar. É importante mencionar que a maneira de definir esta string varia entre bancos diferentes. Você pode explorar mais formas de escrever essas strings acessando ConnectionString;
- Fornecer nome de usuário e senha para nos conectar no banco de dados.

Na figura abaixo, vemos códigos com o passo a passo:

```
// Informações para conexão com o banco de dados.
String stringconexao = "jdbc:postgresql://localhost:5432/meuBD";
String usuario = "postgres";
String senha = "postgre";

try {

    // Carregando o driver do banco de dados.
    Class.forName("org.postgresql.Driver");

    // Criando a conexão com o banco de dados.
    Connection con = DriverManager.getConnection(stringconexao,usuario,senha);
}
catch (Exception ex)
{
    throw new RuntimeException(ex.getMessage());
}
```

Realizada a conexão com o banco de dados, chega o momento de executar as operações de consulta, inserção, atualização e deleção (apagamento) de informações. Temos um conjunto de classes e interfaces já preparados para isso. São elas:

- *Connection* - Representa uma sessão junto ao banco de dados desejado. Vimos a classe no trecho de código do exemplo anterior e vamos executar as instruções SQL dentro da conexão estabelecida.
- *Statement* - Tem como objetivo a execução do comando SQL. Temos também a `PreparedStatement`, que pré-compila o comando e armazena o SQL em um objeto.
- *ResultSet* - Esta interface tem por objetivo armazenar o retorno de uma consulta realizada no banco de dados. As informações das tabelas são recuperadas na sequência e podem ser iteradas em loops para a manipulação.

Vamos agora visualizar a escrita de uma consulta usando os recursos JDBC.

```java
//Criando uma conexão com o banco de dados.
Connection connect = getConnection();

// Criando um objeto que irá conter os comandos SQL.
Statement stmt = connect.createStatement();

// String com o comando SQL a ser executado no banco.
String sql= "SELECT * FROM alunos";

//Como o objetivo é a realização de uma consulta ao banco e dados, vamos pegar,
//o retorno e inicializar um objeto ResultSet, próprio para tratar o retornos de consultas.
ResultSet rs = stmt.executeQuery(sql);

//Percorrendo o objeto e exibindo os dados.
while(rs.next()) {
    System.out.println("#" + rs.getInt("id") +" # "+ rs.getString("nome"));
}

//fechando o objeto.
rs.close();
//fechando o objeto.
stmt.close();

//encerrando a conexão
connect.close();
```

Agora veremos os códigos para a escrita das operações de *insert*, *update* e *delete* . O exemplo mostra um `INSERT`, mas a lógica será a mesma para as demais ações.

```java
//Criando uma conexão com o banco de dados.
Connection connect = getConnection();

// String com o comando SQL a ser executado no banco.
String sql= "INSERT INTO alunos VALUES (?,?,?)";

// Criando um objeto que irá conter os comandos SQL.
PreparedStatement ps = connect.prepareStatement(sql);
ps.setLong(1, 2);
ps.setString(2,"James T. Kirk");
ps.setString(3, "mat03");

//Executando o comando na base de dados.
int resultado = ps.executeUpdate(sql);
if( resultado == 1)
{
    System.out.println("Dados inseridos com sucesso!");
}
//fechando o objeto
ps.close();

//encerrando a conexão
connect.close();
```

No fim, podemos definir os seguintes passos para usar o JDBC:

1. Realizar o carregamento do driver do banco.*Class.forName("org.postgresql.Driver")*.
2. Criar a conexão com o banco. *DriverManager.getConnection(stringconexao,usuario,senha)*.
3. Preparar o comando a ser executado no banco. *String sql="SELECT* FROM alunos"*
4. Executar o comando. Neste ponto, vale a pena ficarmos atentos: quando se trata de uma consulta, usamos *executeQuery*, quando se trata de um insert, update ou delete, usamos *executeUpdate*

5. Tratar o resultado. Quando for o retorno de uma consulta (*ResultSet*), vamos iterar o objeto. Se for o retorno de *insert*, *update* ou *delete*, devemos avaliar o valor retornado.

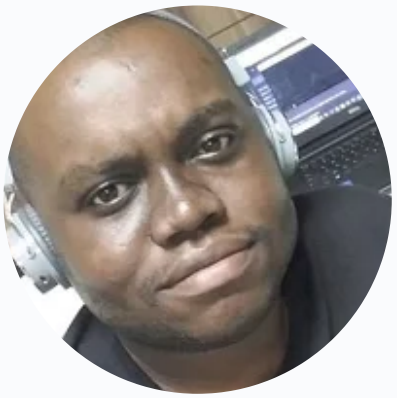
Conclusão

O JDBC é uma API poderosa para manipular base de dados, pois nos permite usar uma estrutura básica para conectar, além de interagir por meio do código Java em diversas fontes de dados, facilitando o trabalho de desenvolvedores Java. Os exemplos deste artigo têm como objetivo mostrar de maneira simples como utilizar a API.

Para aprender sobre JDBC, veja:

- Curso Java e JDBC: Trabalhando com um banco de dados na Alura
- Site ConnectionStrings

Confira neste artigo:



André Bessa

Eu sou programador e instrutor de programação usando C# e .NET. Tenho graduação em Sistemas de Informação e especializações em Engenharia de Software e em Docência Superior. Tenho experiência com desenvolvimento usando Java, PHP, PostgreSQL e MySQL, além de já ter atuado com suporte e implantação de sistemas. Busco sempre aprender mais, também gosto de contribuir com o ensino e divulgação de tecnologia. Nas horas de descanço estou maratonando alguma série ou lendo meus gibis de heróis.

← Artigo Anterior

Strings com JavaScript: o que são e como manipulá-las

Próximo Artigo →

JavaServer Pages: Utilizando os servlets

Veja outros artigos sobre Programação

Artigos

Podcasts

Artigos de educação corporativa

Fale Conosco

Email e telefone

Perguntas frequentes

Novidades e Lançamentos

ENVIAR

CURSOS

Cursos de Programação
Lógica | Python | PHP | Java | .NET | Node JS | C | Computação | Jogos | IoT

Cursos de Front-end
HTML, CSS | React | Angular | JavaScript | jQuery

Cursos de Data Science