Programação Avançada - AULA 12

Matheus Moresco

Análise e Desenvolvimento de Sistemas - 5º Período

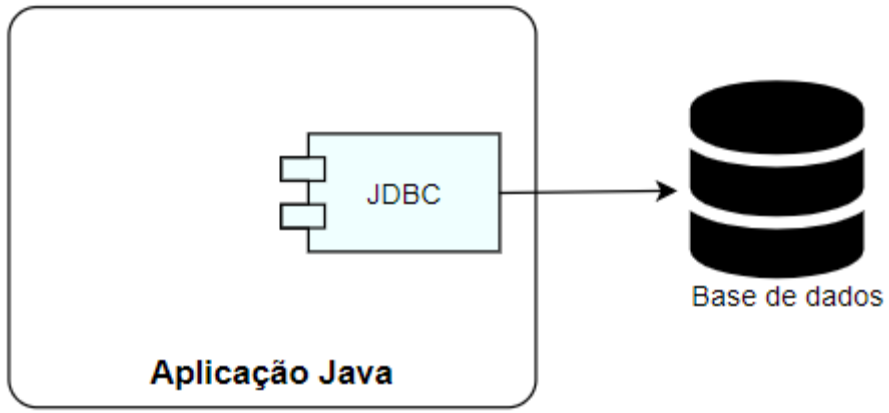
2025/01

Introdução

- Persistência de dados com JDBC
  - O que é o JDBC?
  - Pra que serve?
  - Como usar?
  - Fazendo um CRUD com JDBC

O que é o JDBC?

- JDBC (**Java Database Connectivity**) é uma **API Java** que permite a comunicação entre aplicações Java e bancos de dados relacionais.
- Permite executar comandos SQL dentro de aplicações Java

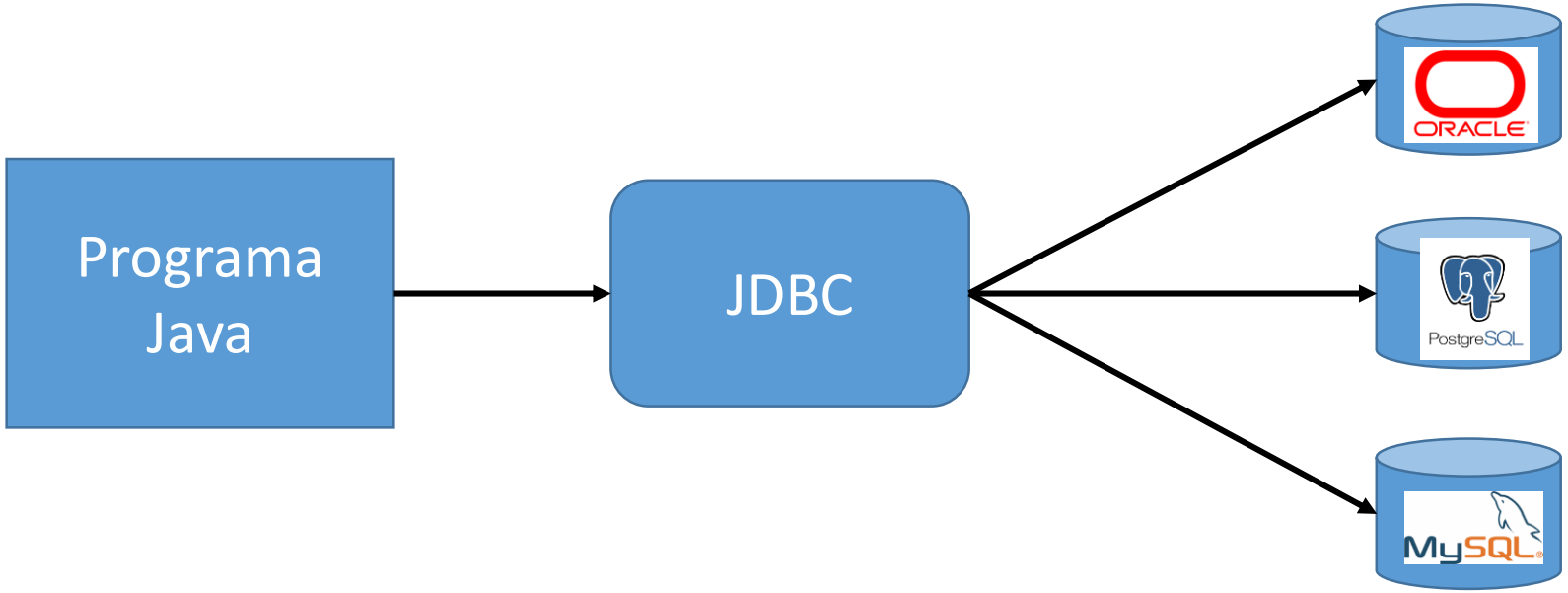


Principais características do JDBC

- Facilita a execução de comandos SQL (SELECT, INSERT, UPDATE, DELETE) diretamente no Java
- Fornece uma interface padronizada para conectar diferentes bancos de dados (MySQL, PostgreSQL, Oracle, etc.)
- Permite transações e gerenciamento de conexões
- Suporta drivers para diversos bancos

Por que usar JDBC?

- Conexão direta com diferentes bancos de dados (MySQL, PostgreSQL, Oracle, etc.)
- Permite manipulação estruturada de dados
- Integração com frameworks ORM como Hibernate e JPA



Como Funciona?

- O JDBC (Java Database Connectivity) funciona como um intermediário entre uma aplicação Java e um banco de dados relacional. Ele realiza a interpretação e a conversão dos comandos necessários para a manipulação do banco de dados, de acordo com as características de cada um.
- Isso porque, apesar de todos os bancos de dados relacionais utilizarem a linguagem SQL, existem diferenças entre as diversas bases existentes no mercado.
- É preciso utilizar o driver JDBC de acordo com a base de dados utilizada, ou seja, se formos utilizar um banco MySQL, devemos usar o JDBC apropriado para ele.
- Devemos passar os dados necessários para fazer a conexão com o banco e as instruções SQL.

Arquitetura do JDBC

- **Driver JDBC** – É um software que atua como um **conector** entre a aplicação Java e o banco de dados específico.
- **Connection** – Representa uma **sessão ativa** com o banco de dados. Gerencia a comunicação entre a aplicação e o banco.
- **Statement/PreparedStatement** – São objetos usados para **executar comandos SQL** no banco.
- **ResultSet** – É um objeto que **armazena os resultados de uma consulta SQL**.

Fluxo de funcionamento do JDBC

1. **Carregamento do Driver JDBC**
   - O driver JDBC é um conector que permite que Java se comunique com o banco de dados específico.
2. **Estabelecimento da Conexão**
   - A aplicação usa a classe *DriverManager* para criar uma conexão com o banco.
3. **Execução de Comandos SQL**
   - Usamos *Statement* ou *PreparedStatement* para executar consultas (SELECT) ou modificações (INSERT, UPDATE, DELETE).
4. **Processamento dos Resultados**
   - Quando uma consulta retorna dados (SELECT), o resultado é armazenado em um ResultSet e pode ser percorrido.
5. **Fechamento da Conexão**
   - Para evitar desperdício de recursos, a conexão e os objetos relacionados devem ser fechados após o uso.

Configuração do JDBC

Passos para configurar JDBC

1. Baixar e adicionar o driver JDBC do banco escolhido (exemplo: MySQL)
2. Criar um banco de dados e tabela de exemplo
3. Estabelecer uma conexão a partir do Java

```java
import java.sql.Connection;
import java.sql.DriverManager;

public class App {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) throws Exception {
        String url = "jdbc:mysql://localhost:3306/teste";
        String username = "root";
        String password = "root";
        Connection conexao = DriverManager.getConnection(url, username, password);

        conexao.close();

    }
}
```

Operações CRUD com JDBC

CRUD representa um conjunto de operações básicas para se trabalhas com a manipulação de dados. As operações de um CRUD são:

- **CREATE**: Criação de novas instâncias.
- **READ**: Consultar os dados salvos no banco.
- **UPADTE**: Atualizar/Editar os dados que já existem no banco.
- **DELETE**: Deletar dados da base.

CREATE - Inserindo dados

- Comando em SQL:

```sql
INSERT INTO usuarios (nome, email) VALUES ('João Silva', 'joao@email.com');
```

- Usando JDBC :

```java
String sql = "INSERT INTO usuarios (nome, email) VALUES (?, ?)";

try (Connection conexao = DriverManager.getConnection(url, username, password);
    PreparedStatement stmt = conexao.prepareStatement(sql)) {
    stmt.setString(parameterIndex:1, x:"Maria Souza");
    stmt.setString(parameterIndex:2, x:"maria@email.com");
    stmt.executeUpdate();

    System.out.println(x:"Usuário inserido com sucesso!");
} catch (SQLException e) {
    e.printStackTrace();
}
```

READ- Consultando os dados

- Comando em SQL:

```sql
SELECT * FROM usuarios;
```

- Usando JDBC :

```java
String sql = "SELECT * FROM usuarios";

try (Connection conexao = DriverManager.getConnection(url, usuario, senha);
     Statement stmt = conexao.createStatement();
     ResultSet rs = stmt.executeQuery(sql)) {

    while (rs.next()) {
        System.out.println(
            "ID: " + rs.getInt(columnLabel:"id") +
            ", Nome: " + rs.getString(columnLabel:"nome") +
            ", Email: " + rs.getString(columnLabel:"email"));
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

UPDATE – Editando dados

- Comando em SQL:

```sql
UPDATE usuarios SET nome = 'João Pedro' WHERE id = 1;
```

- Usando JDBC :

```java
String sql = "UPDATE usuarios SET nome = ? WHERE id = ?";
try (Connection conexao = DriverManager.getConnection(url, usuario, senha);
     PreparedStatement stmt = conexao.prepareStatement(sql)) {

    stmt.setString(parameterIndex:1, x:"João Pedro");
    stmt.setInt(parameterIndex:2, x:1);
    stmt.executeUpdate();

    System.out.println(x:"Usuário atualizado com sucesso!");
} catch (SQLException e) {
    e.printStackTrace();
}
```

DELETE – Excluindo dados

- Comando em SQL:

```sql
DELETE FROM usuarios WHERE id = 1;
```

- Usando JDBC :

```java
String sql = "DELETE FROM usuarios WHERE id = ?";

try (Connection conexao = DriverManager.getConnection(url, usuario, senha);
     PreparedStatement stmt = conexao.prepareStatement(sql)) {

    stmt.setInt(parameterIndex:1, x:1);
    stmt.executeUpdate();

    System.out.println(x:"Usuário deletado com sucesso!");
} catch (SQLException e) {
    e.printStackTrace();
}
```

Boas Práticas e Segurança no JDBC

- Usar PreparedStatement para evitar SQL Injection
- Fechar conexões, statements e result sets corretamente
- Evitar consultas desnecessárias e otimizar SQL

Exemplo Prático

1. Criar um dataset no mysql.
2. Fazer uma conexão com o dataset usando o JDBC.
3. Criar a tabela de Pessoa no banco e criar uma classe correspondente no Java.
4. Fazer o CRUD do objeto de Pessoa usando JDBC.