📚 Conectividade com Banco de Dados (BD) em Java

🔗 14.1. Java DataBase Connectivity (JDBC API)

- **JDBC** é a API padrão do Java para se comunicar com **bancos de dados relacionais**.
- Permite escrever código independente do banco de dados (portável).
- Fornecedores como Oracle, MySQL, PostgreSQL oferecem **drivers JDBC específicos**.

🧱 Arquitetura JDBC

Inclui:

- **Driver Manager**: carrega o driver correto.
- **Driver JDBC**: responsável pela comunicação com o banco.
- **Connection, Statement, ResultSet**: principais interfaces/classes usadas para interação.

📦 14.1.2. Tipos de Drivers JDBC

Tipo	Descrição	Portabilidade	Status Atual
Tipo 1	JDBC-ODBC Bridge (usa ODBC)	❌ Baixa	**Obsoleto** desde Java 8
Tipo 2	Parcialmente Java + código nativo	❌ Baixa	Pouco usado
Tipo 3	Totalmente Java, usa middleware	✅ Alta	Raro hoje
Tipo 4	Totalmente Java, comunicação direta com o BD	✅ **Alta**	✅ **Mais comum**

🛠️ Exemplo Básico de Uso do JDBC (Tipo 4)

```
import java.sql.*;
```

```
public class ConexaoExemplo {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/meubanco";
        String usuario = "root";
        String senha = "senha";

        try (Connection conn = DriverManager.getConnection(url, usuario, senha)) {
            System.out.println("Conexão bem-sucedida!");

            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM clientes");

            while (rs.next()) {
                System.out.println(rs.getString("nome"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

✅ O que é JDBC?

JDBC (Java Database Connectivity) é uma API da própria JDK que permite conectar uma aplicação Java a bancos de dados relacionais usando comandos SQL.

✅ Principais interfaces da API JDBC

Interface / Classe	Função
`DriverManager`	Gerencia os drivers e cria conexões com o banco de dados
`Connection`	Representa uma conexão ativa com o banco
`Statement` / `PreparedStatement`	Executa comandos SQL (com ou sem parâmetros)
`ResultSet`	Armazena os resultados de uma consulta SQL

✅ Passos básicos para usar JDBC na JDK 7

1. Carregar o driver JDBC

Cada banco tem seu driver (ex: MySQL usa `com.mysql.cj.jdbc.Driver`).

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

Esse comando **registra o driver** para que o `DriverManager` saiba como se conectar ao banco.

---

2. Estabelecer a conexão com o banco de dados

Você usa o `DriverManager.getConnection()` passando a **URL de conexão**, usuário e senha.

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/agenda", "root", "senha");
```

---

3. Executar comandos SQL com `PreparedStatement`

- **Comando SQL com parâmetros**: Usa ? como marcador.
- **Você define os valores depois com `setString`, `setInt`, etc.**

```
String sql = "INSERT INTO contatos (nome, email, telefone) VALUES (?, ?, ?)";
PreparedStatement stmt = con.prepareStatement(sql);
stmt.setString(1, "João");
stmt.setString(2, "joao@email.com");
stmt.setString(3, "99999-9999");
stmt.executeUpdate(); // Para INSERT, UPDATE ou DELETE
```

---

4. Consultar dados com `ResultSet`

```
String sql = "SELECT * FROM contatos";
PreparedStatement stmt = con.prepareStatement(sql);
ResultSet rs = stmt.executeQuery();

while (rs.next()) {
    int id = rs.getInt("id");
    String nome = rs.getString("nome");
    String email = rs.getString("email");
    String telefone = rs.getString("telefone");
    System.out.println(nome + " - " + email);
}
```

---

5. Fechar todos os recursos (boa prática)

```
rs.close();
stmt.close();
con.close();
```

Na JDK 7, você pode usar **try-with-resources** (também chamado de try automático):

```
try (Connection con = DriverManager.getConnection(...);
     PreparedStatement stmt = con.prepareStatement(...)) {

    // execução aqui
}
```

✅ Exemplo completo: Inserir um contato

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

public class InserirContato {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            try (Connection con = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/agenda", "root", "senha")) {

                String sql = "INSERT INTO contatos (nome, email, telefone) VALUES (?, ?, ?)";
                try (PreparedStatement stmt = con.prepareStatement(sql)) {
                    stmt.setString(1, "Maria");
                    stmt.setString(2, "maria@email.com");
                    stmt.setString(3, "88888-8888");
                    stmt.executeUpdate();
                    System.out.println("Contato inserido com sucesso!");
                }

            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

✅ Resumo visual

```
Class.forName()       -> Carrega o driver
DriverManager         -> Cria a conexão
Connection            -> Representa a conexão ativa
PreparedStatement     -> Executa comandos SQL com segurança
ResultSet             -> Retorna os dados da consulta
```

✅ Estrutura do banco `agenda`

```
CREATE TABLE contatos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100),
    telefone VARCHAR(20)
);
```

✅ Classe `Contatos`

```
public class Contatos {
    private int id;
    private String nome;
    private String email;
    private String telefone;

    // Construtores
    public Contatos() {}

    public Contatos(String nome, String email, String telefone) {
        this.nome = nome;
        this.email = email;
        this.telefone = telefone;
    }

    public Contatos(int id, String nome, String email, String telefone) {
        this.id = id;
        this.nome = nome;
        this.email = email;
        this.telefone = telefone;
    }

    // Getters e Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getTelefone() { return telefone; }
    public void setTelefone(String telefone) { this.telefone = telefone; }
```

}

✅ Classe `DaoContatos` com CRUD

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class DaoContatos {

    private Connection abrirConexao() throws Exception {
        Class.forName("com.mysql.cj.jdbc.Driver");
        return DriverManager.getConnection("jdbc:mysql://localhost:3306/agenda", "root",
"senha");
    }

    public void incluir(Contatos contato) throws Exception {
        String sql = "INSERT INTO contatos (nome, email, telefone) VALUES (?, ?, ?)";
        try (Connection con = abrirConexao();
             PreparedStatement stmt = con.prepareStatement(sql)) {
            stmt.setString(1, contato.getNome());
            stmt.setString(2, contato.getEmail());
            stmt.setString(3, contato.getTelefone());
            stmt.executeUpdate();
        }
    }

    public void remover(int id) throws Exception {
        String sql = "DELETE FROM contatos WHERE id = ?";
        try (Connection con = abrirConexao();
             PreparedStatement stmt = con.prepareStatement(sql)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        }
    }

    public Contatos buscar(int id) throws Exception {
        String sql = "SELECT * FROM contatos WHERE id = ?";
        try (Connection con = abrirConexao();
             PreparedStatement stmt = con.prepareStatement(sql)) {
            stmt.setInt(1, id);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new Contatos(
                        rs.getInt("id"),
```

```
                    rs.getString("nome"),
                    rs.getString("email"),
                    rs.getString("telefone")
                );
            }
            return null;
        }
    }
}

public void alterar(Contatos contato) throws Exception {
    String sql = "UPDATE contatos SET nome = ?, email = ?, telefone = ? WHERE id = ?";
    try (Connection con = abrirConexao();
         PreparedStatement stmt = con.prepareStatement(sql)) {
        stmt.setString(1, contato.getNome());
        stmt.setString(2, contato.getEmail());
        stmt.setString(3, contato.getTelefone());
        stmt.setInt(4, contato.getId());
        stmt.executeUpdate();
    }
}

public List<Contatos> listar() throws Exception {
    List<Contatos> lista = new ArrayList<>();
    String sql = "SELECT * FROM contatos";
    try (Connection con = abrirConexao();
         PreparedStatement stmt = con.prepareStatement(sql);
         ResultSet rs = stmt.executeQuery()) {

        while (rs.next()) {
            Contatos contato = new Contatos(
                rs.getInt("id"),
                rs.getString("nome"),
                rs.getString("email"),
                rs.getString("telefone")
            );
            lista.add(contato);
        }
    }
    return lista;
}
}
```

✅ Classe `TesteBancoDados`

```
import java.util.List;

public class TesteBancoDados {
    public static void main(String[] args) {
        DaoContatos dao = new DaoContatos();

        try {
            // Incluir
            Contatos novo = new Contatos("Ana", "ana@email.com", "77777-7777");
            dao.incluir(novo);

            // Listar
            List<Contatos> contatos = dao.listar();
            for (Contatos c : contatos) {
                System.out.println(c.getId() + " - " + c.getNome() + " - " + c.getEmail() + " - " +
c.getTelefone());
            }

            // Buscar
            Contatos buscado = dao.buscar(1);
            if (buscado != null) {
                System.out.println("Encontrado: " + buscado.getNome());
            }

            // Alterar
            if (buscado != null) {
                buscado.setNome("Ana Maria");
                dao.alterar(buscado);
            }

            // Remover
            dao.remover(2); // exemplo

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

✅ 15.1.4 – Hierarquia de classes JDBC

JDBC é estruturado com interfaces e implementações concretas fornecidas pelo driver do banco de dados.

As **principais interfaces** na hierarquia do JDBC são:

```
java.sql.Driver
  ↳ java.sql.Connection
    ↳ java.sql.Statement
      ↳ java.sql.PreparedStatement
    ↳ java.sql.CallableStatement
  ↳ java.sql.ResultSet
```

Como funciona:

- `DriverManager` gerencia os **drivers de banco**.
- `Connection` representa a **conexão aberta com o banco**.
- `PreparedStatement` permite executar **SQL com parâmetros** de forma segura (evita SQL Injection).
- `ResultSet` representa o **resultado de uma consulta SELECT**, permitindo iterar linha por linha.

✅ 15.1.5 – Métodos do PreparedStatement e ResultSet

▸ PreparedStatement

Usado para executar comandos SQL com parâmetros (`?`).

Exemplo:

```
PreparedStatement stmt = conexao.prepareStatement(
   "INSERT INTO contatos (nome, telefone, email, data_cadastro) VALUES (?, ?, ?, ?)"
);
stmt.setString(1, contato.getNome());
stmt.setString(2, contato.getTelefone());
stmt.setString(3, contato.getEmail());
stmt.setDate(4, new java.sql.Date(contato.getData().getTime()));
```

Métodos comuns:

- `setString(int pos, String valor)`
- `setInt(int pos, int valor)`
- `setDate(int pos, java.sql.Date valor)`
- `setLong`, `setBoolean`, `setDouble`, etc.

▸ ResultSet

É o objeto que armazena os resultados de uma consulta (SELECT).

Exemplo:

```
ResultSet rs = stmt.executeQuery();
while (rs.next()) {
    String nome = rs.getString("nome");
    String telefone = rs.getString("telefone");
}
```

Métodos comuns:

- `getString(String coluna)`
- `getInt(String coluna)`
- `getDate(String coluna)`
- `getBoolean`, `getLong`, `getDouble`, etc.

✅ Conversão entre tipos SQL e Java (Quadro 23)

Tipo SQL	Tipo Java correspondente
CHAR, VARCHAR	`String`
NUMERIC, DECIMAL	`BigDecimal`
TINYINT	`boolean` ou `byte`

SMALLINT	`short`
INTEGER	`int`
BIGINT	`long`
FLOAT, REAL	`float`
DOUBLE	`double`
DATE	`java.sql.Date`
TIME	`java.sql.Time`
TIMESTAMP	`java.sql.Timestamp`
BINARY	`byte[]`