

# Persistência de Dados

**Java JDBC**

**Java DB (Derby)**

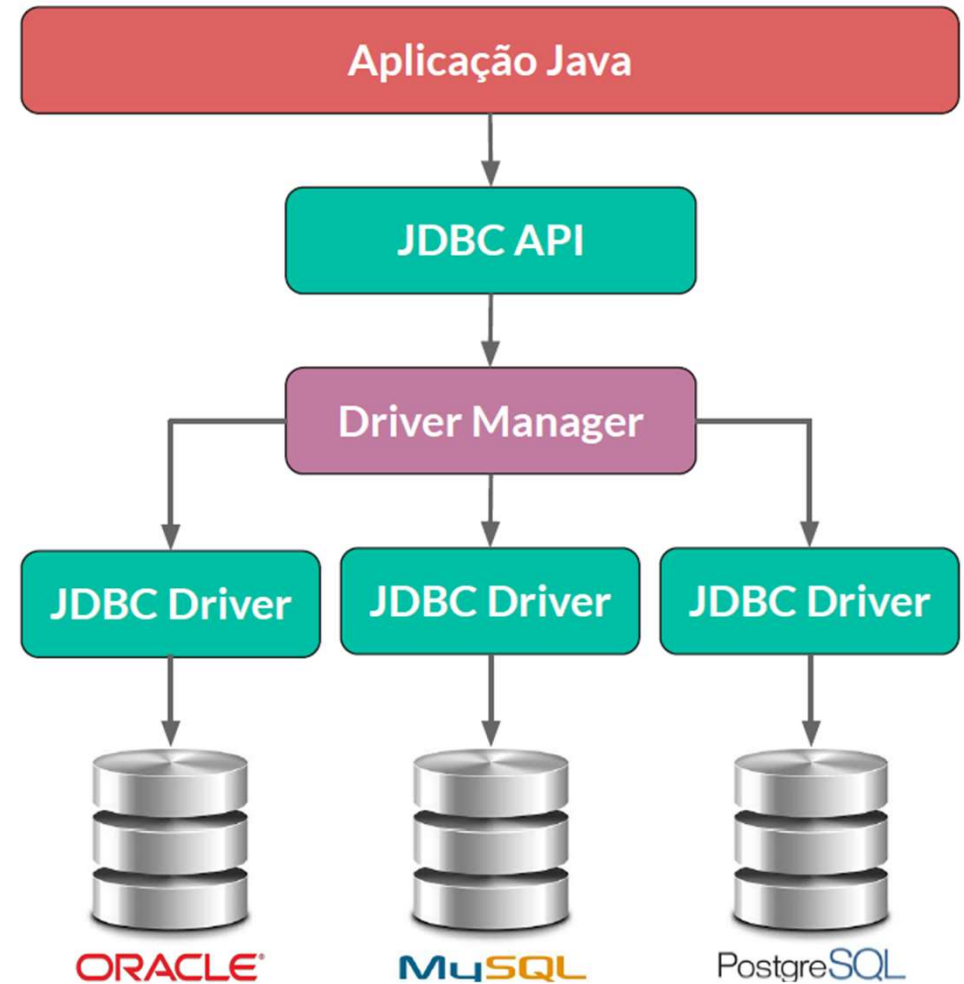
**Java DAO**

# JDBC

## (Java Database Connectivity)

### O que é?

- Uma biblioteca;
- Implementada em Java;
- Disponibiliza classes e interfaces para acesso a banco de dados;





# JDBC

- Principais classes e interfaces do pacote *java.sql*
  - **DriverManager**, cria conexão com o banco de dados;
  - **Connection**, mantém uma conexão aberta com o banco;
  - **Statement**, gerencia e executa instruções SQL;
  - **PreparedStatement**, gerencia e executa instruções SQL (parâmetros);
  - **ResultSet**, recebe os dados obtidos em uma pesquisa ao banco

# Persistência de Dados

- **Java DB (Derby)**

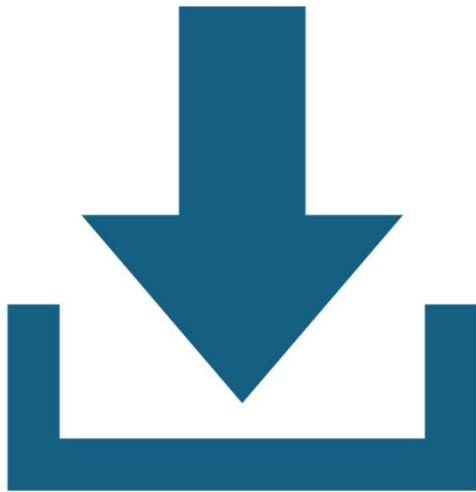
- É uma distribuição da Sun com suporte do Apache Derby.
- O **Java DB** é um servidor de banco de dados, escrito inteiramente em Java, com suporte a SQL, JDBC API e à tecnologia Java EE.



# JAVA DB (Derby)



- Site: <https://db.apache.org/derby/>
- Documentação: <https://db.apache.org/derby/manuals/index.html>
- Download: [https://db.apache.org/derby/derby\\_downloads.html](https://db.apache.org/derby/derby_downloads.html)



# Instalando o JAVA DB

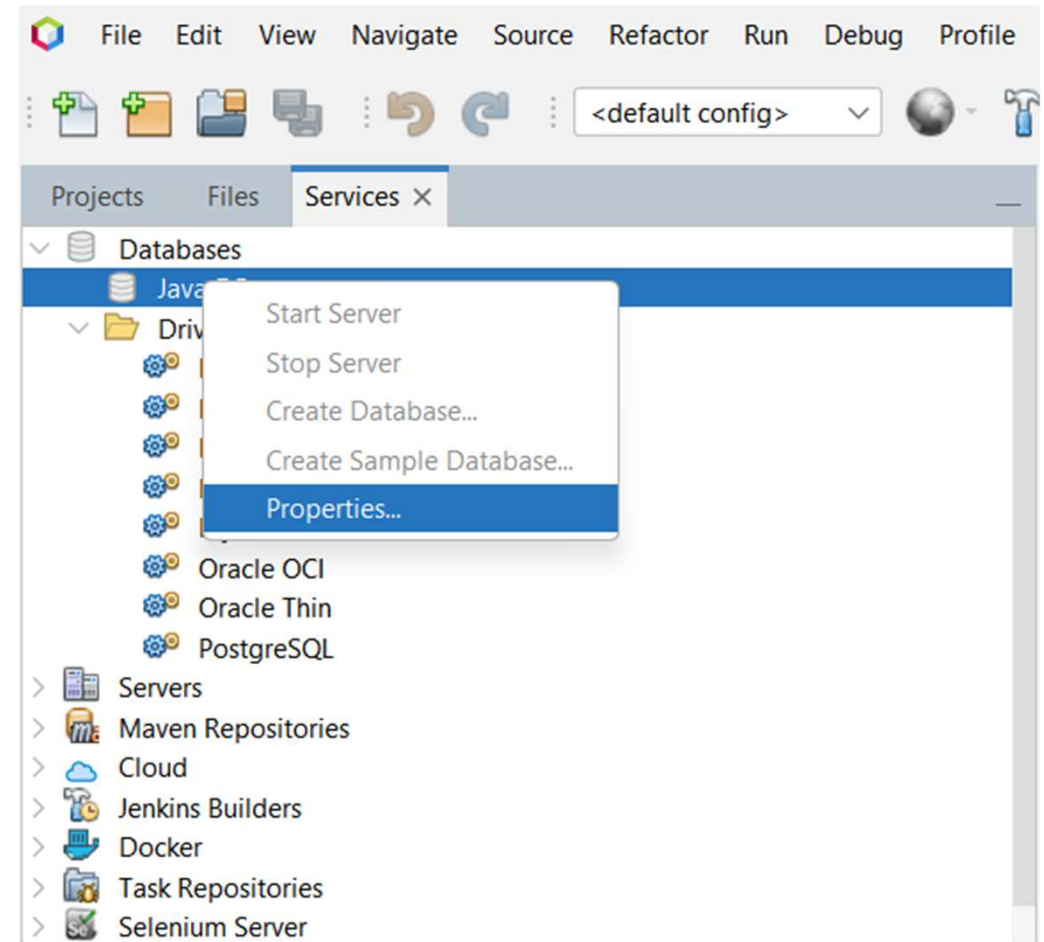
- Faça download do arquivo *bin.zip* da versão correspondente ao Java utilizado.

Para o Java 21: <https://dlcdn.apache.org/db/derby/db-derby-10.17.1.0/db-derby-10.17.1.0-bin.zip>

- Extraia o arquivo na pasta de instalação do JAVA ou na pasta do projeto.
- Abra o **Netbeans** e na aba **Window**, selecione **Services**.

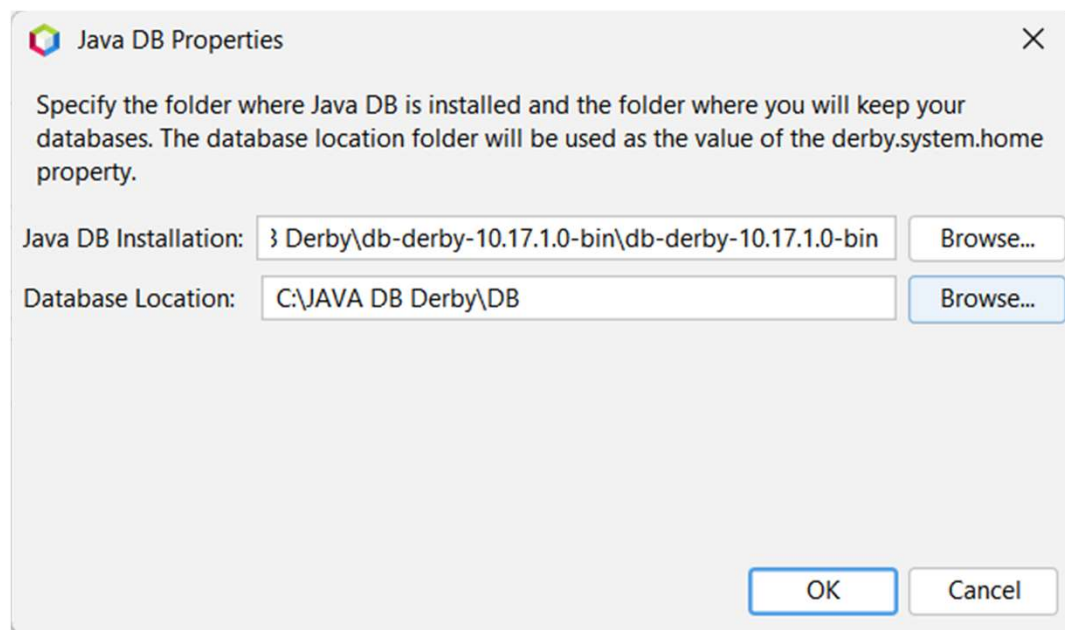
# Instalando o JAVA DB

Na aba Services:  
Localize Java DB → Properties..



# Instalando o JAVA DB

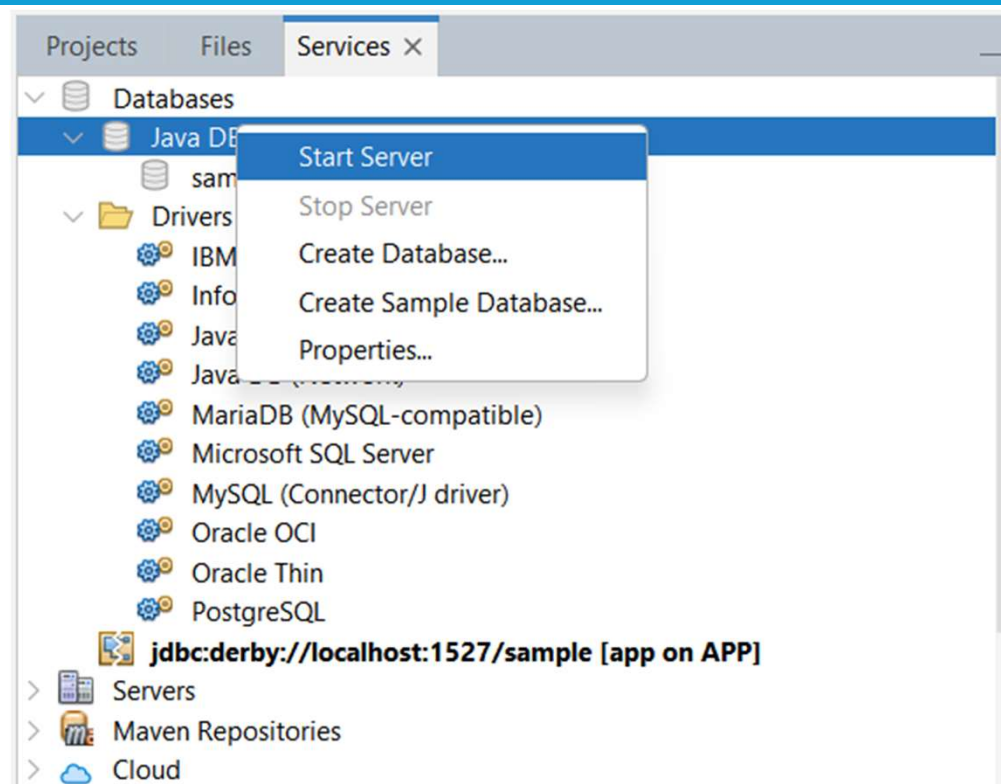
- Selecione o pasta com o Java DB descompactado.
- E a pasta onde será salvo o arquivo do banco de dados





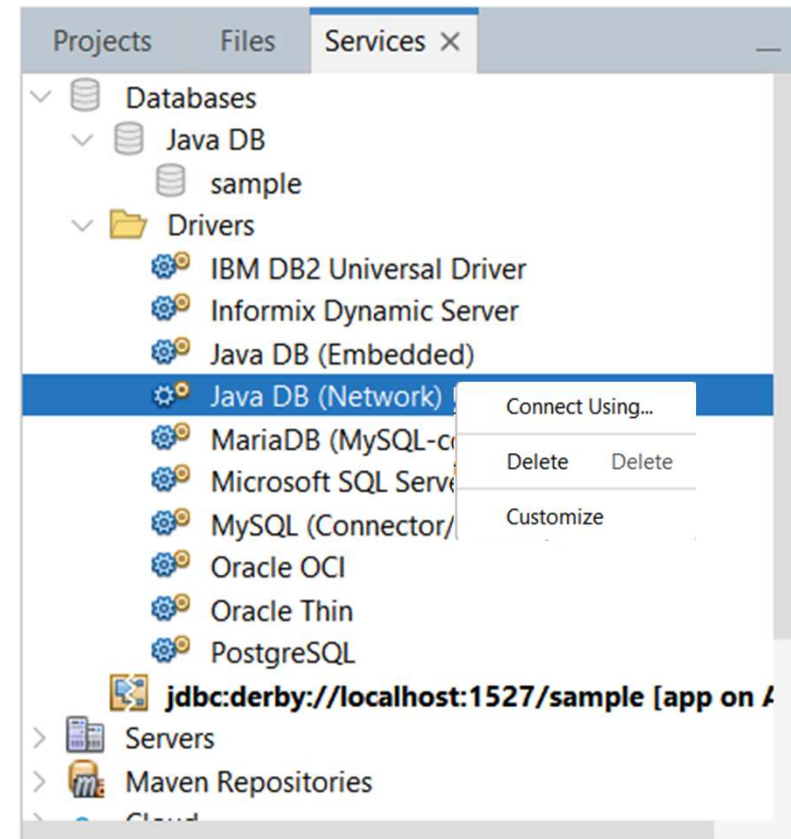
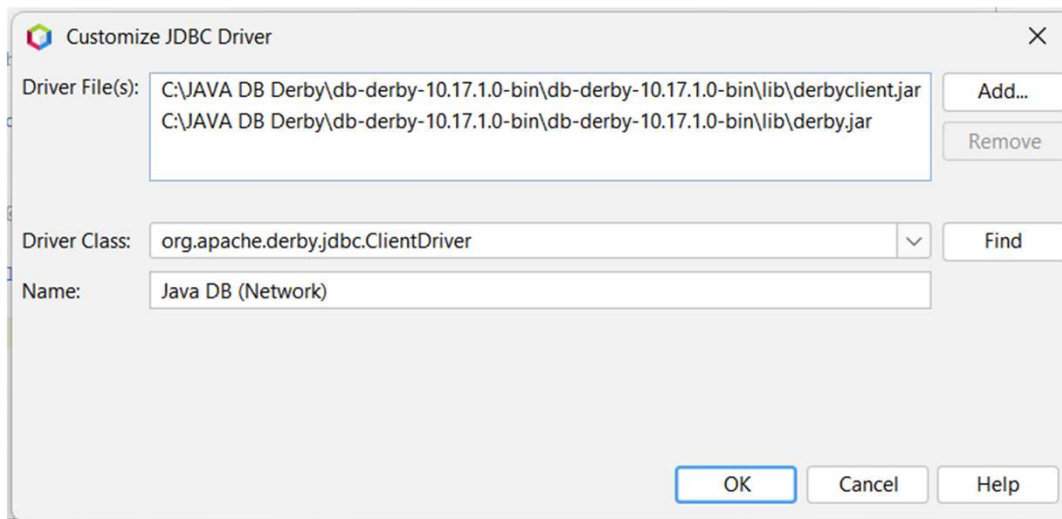
# Instalando o JAVA DB

- Inicialize o servidor do JAVA DB.
- No console de saída verifique se o servidor foi inicializado: Sun Apr 07 11:10:18 BRT 2024 :  
Apache Derby Servidor de Rede - 10.17.1.0 - (1913217)  
iniciado e pronto para aceitar conexões na porta 1527 em {3}



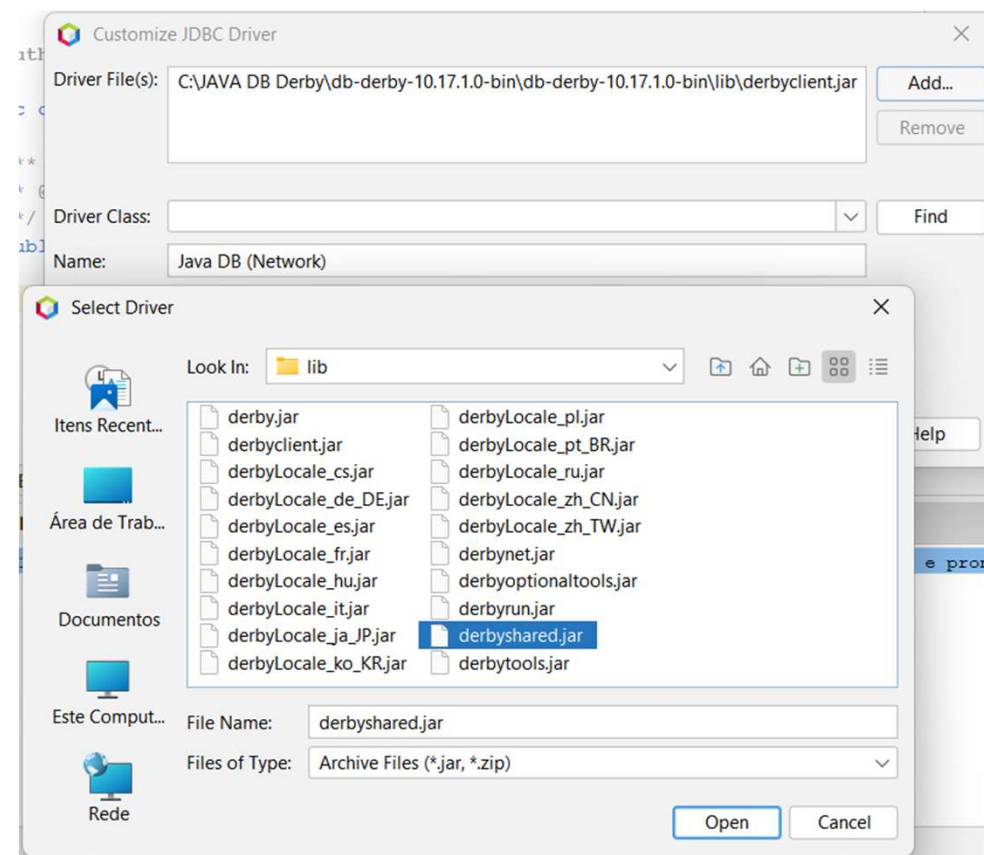
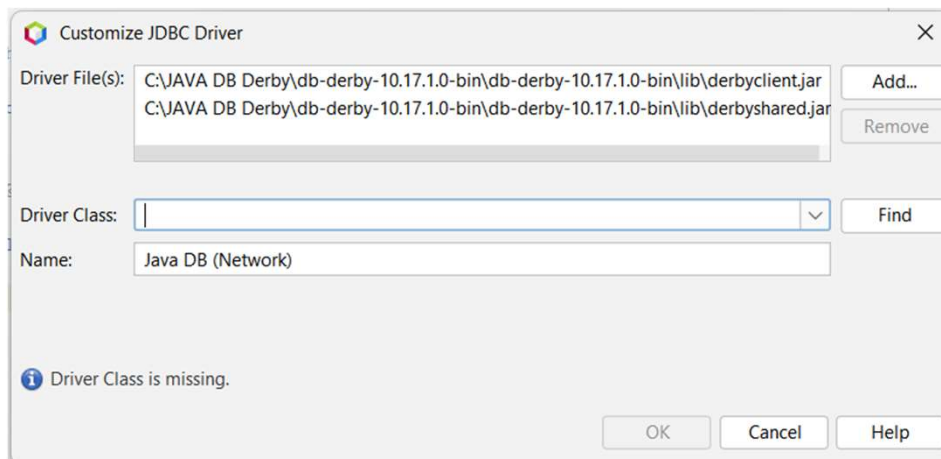
# Configurando uma nova conexão

- Procure Java DB (Network) → Customize.
- **Remova derby.jar**



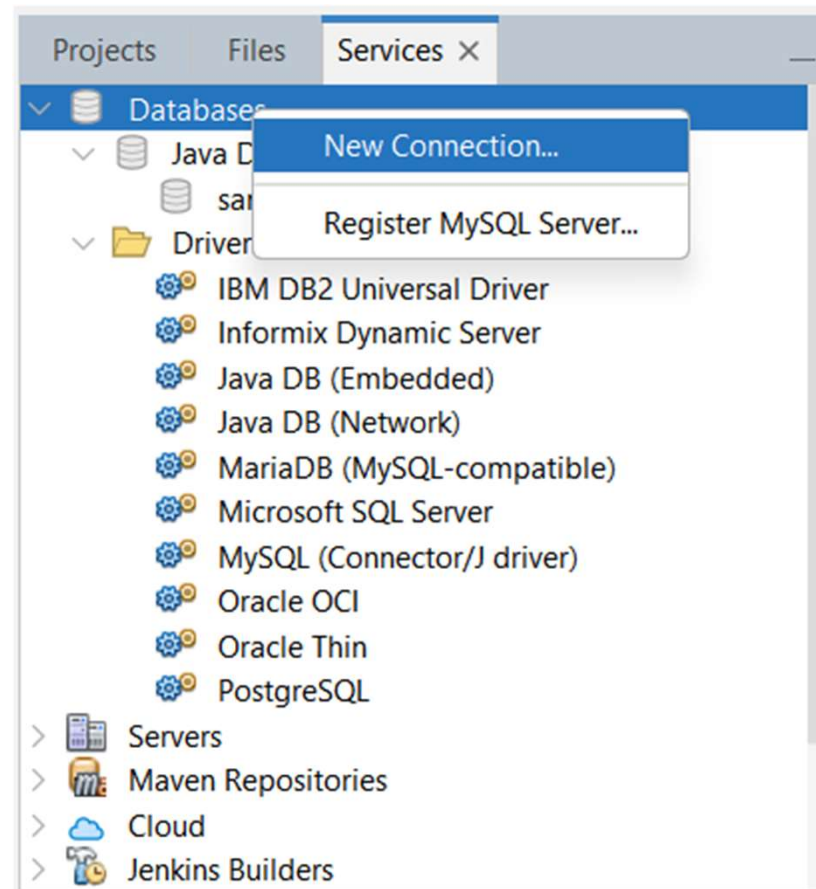
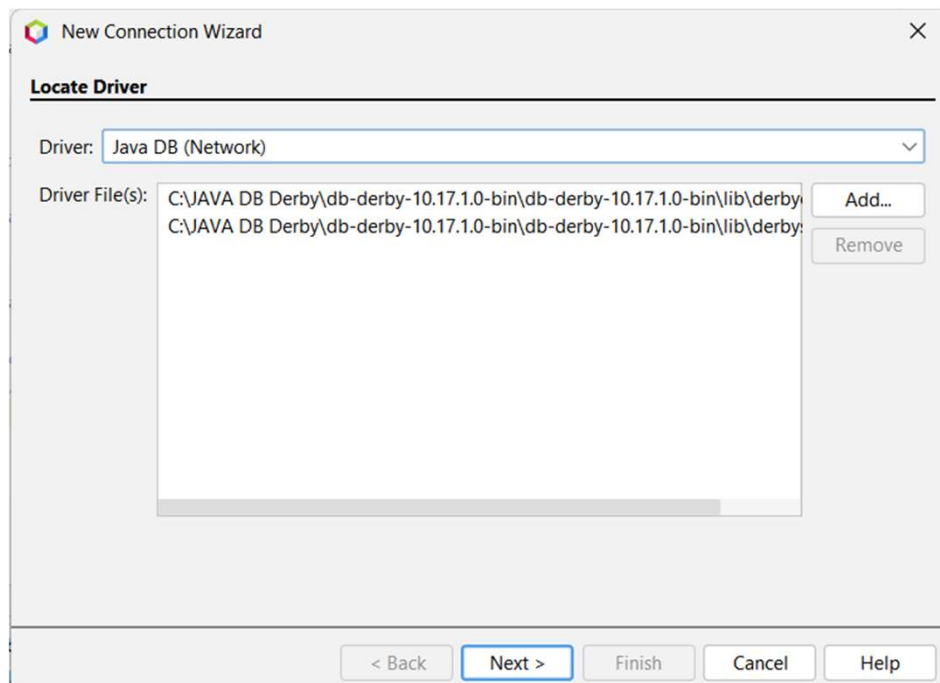
# Configurando uma nova conexão

- Adicione **derbyshared.jar**, localizado na pasta **lib** do pacote de instalação do JAVA DB.
- Utilize o botão *Find* para localizar o *Driver Class*.



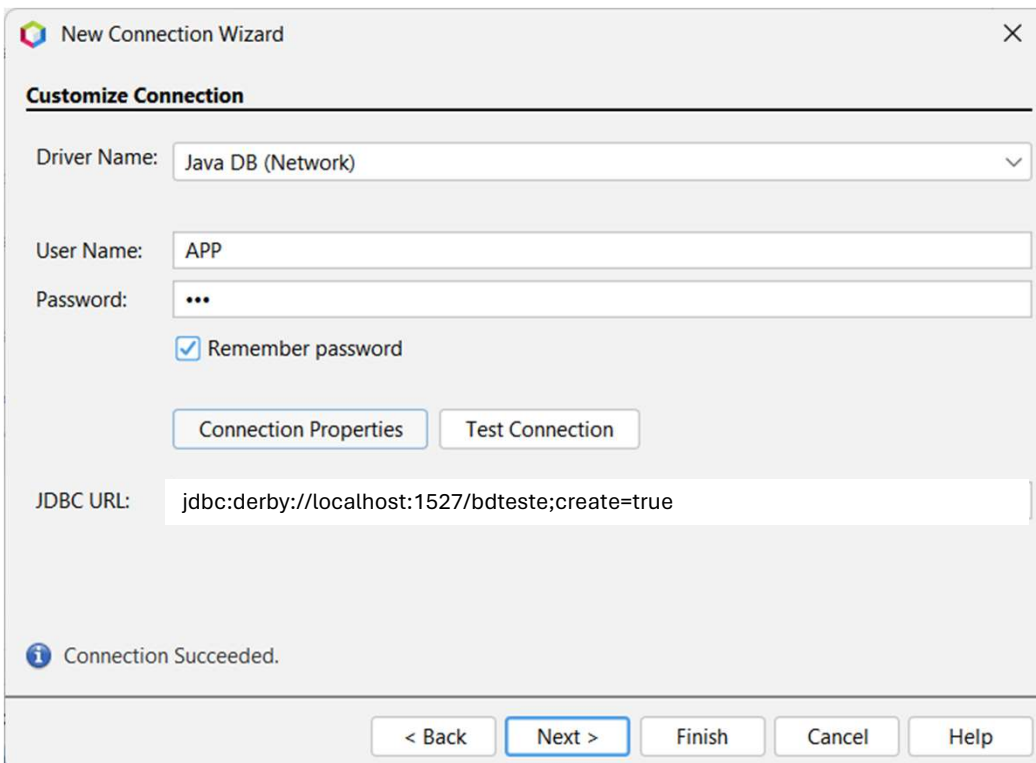
# Criando uma nova conexão

- *Selecione Java DB (Network).*



# Criando uma nova conexão

- Configure **UserName(APP)** e **Password(123)**.
- **JDBC URL:**  
`jdbc:derby://localhost:1527/bdteste;create=true`



The screenshot shows the 'New Connection Wizard' dialog box, specifically the 'Customize Connection' step. The 'Driver Name' is set to 'Java DB (Network)'. The 'User Name' is 'APP' and the 'Password' is masked with three dots. The 'Remember password' checkbox is checked. There are buttons for 'Connection Properties' and 'Test Connection'. The 'JDBC URL' is 'jdbc:derby://localhost:1527/bdteste;create=true'. At the bottom, a status bar indicates 'Connection Succeeded.' and navigation buttons include '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

New Connection Wizard

**Customize Connection**

Driver Name: Java DB (Network)

User Name: APP

Password: ...

☒ Remember password

Connection Properties Test Connection

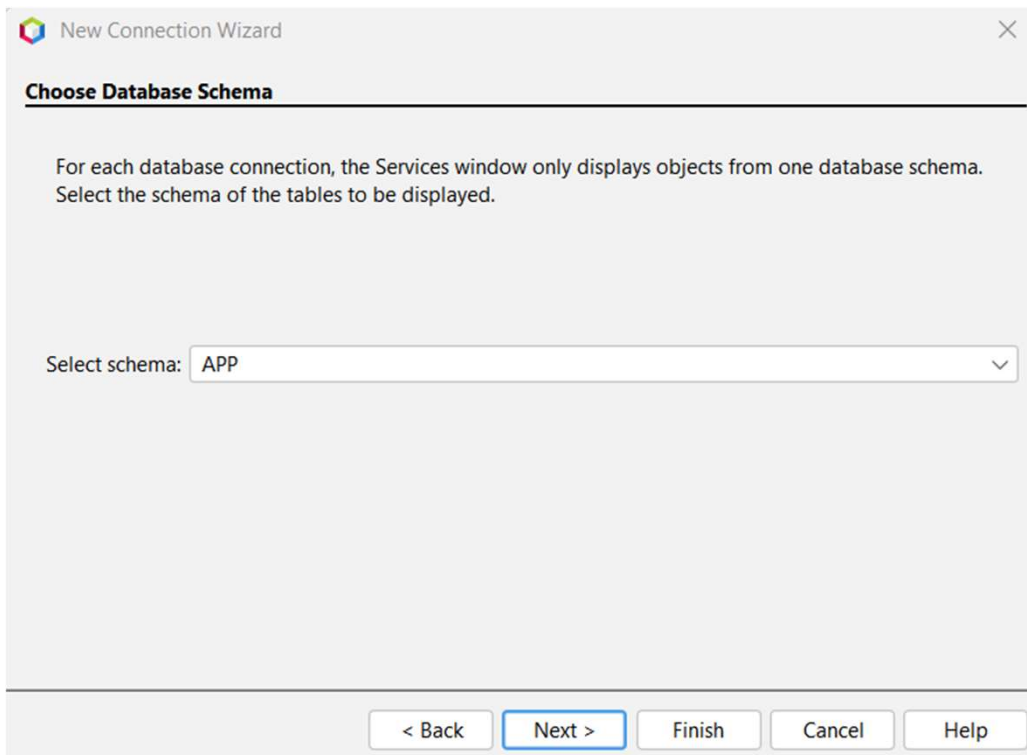
JDBC URL: jdbc:derby://localhost:1527/bdteste;create=true

Connection Succeeded.

< Back Next > Finish Cancel Help

# Criando uma nova conexão

- Finalize a configuração da conexão.



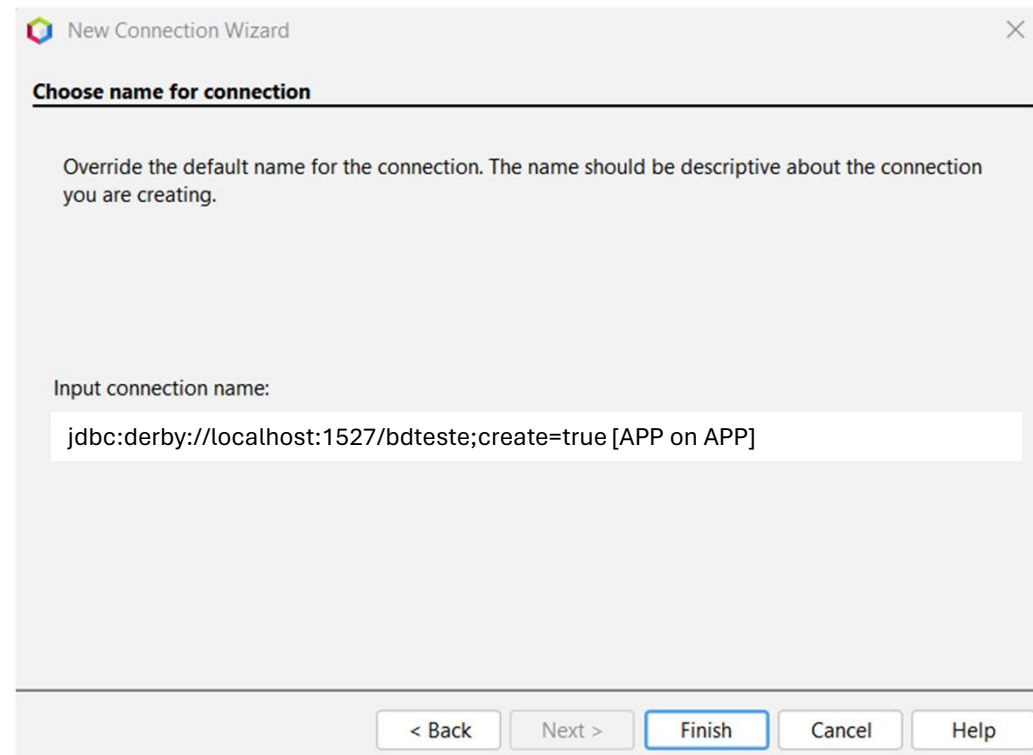
New Connection Wizard

**Choose Database Schema**

For each database connection, the Services window only displays objects from one database schema. Select the schema of the tables to be displayed.

Select schema:

< Back   Next >   Finish   Cancel   Help



New Connection Wizard

**Choose name for connection**

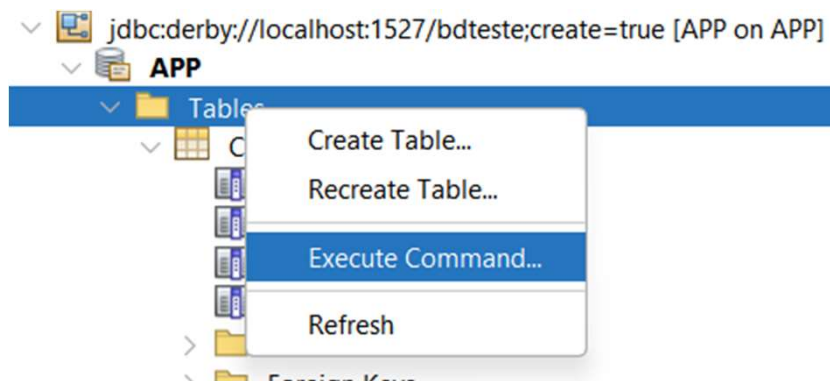
Override the default name for the connection. The name should be descriptive about the connection you are creating.

Input connection name:

< Back   Next >   Finish   Cancel   Help

# Criando uma nova tabela

- As tabelas podem ser criadas através do botão **Create Table** ou por **linha de comando**.
- Selecione **Execute Command...** e copie as instruções para a criação da tabela veículo.
- Execute **Run SQL (Ctrl+Shift+E)**

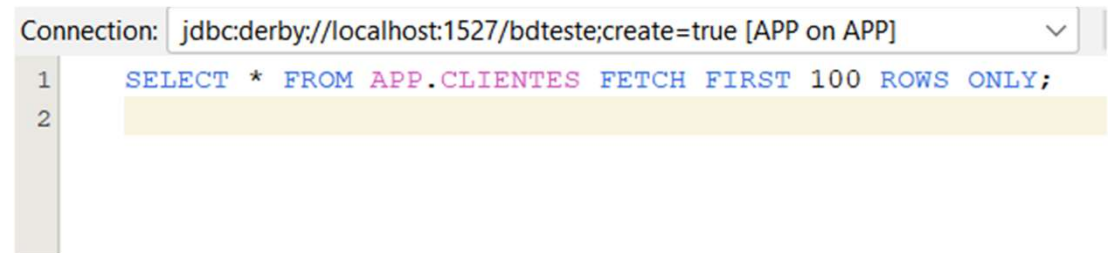
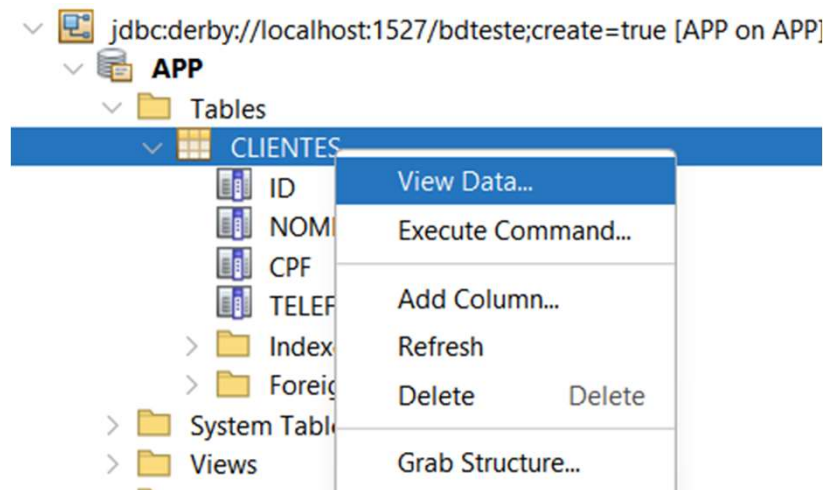


```
CREATE TABLE clientes (  
    id INTEGER NOT NULL GENERATED ALWAYS  
        AS IDENTITY (START WITH 1, INCREMENT BY 1),  
    nome VARCHAR(50),  
    cpf VARCHAR(14),  
    telefone VARCHAR(14)  
);  
  
INSERT INTO CLIENTES (nome, cpf, telefone)  
VALUES ('Cliente 1', '111.111.111-11', '(11) 1111-1111');  
  
INSERT INTO CLIENTES (nome, cpf, telefone)  
VALUES ('Cliente 2', '222.222.222-22', '(22) 2222-2222');  
  
INSERT INTO CLIENTES (nome, cpf, telefone)  
VALUES ('Cliente 3', '333.333.333-33', '(33) 3333-3333');
```



# Criando uma nova tabela

- Para visualizar os dados da tabela.



Max. rows: 100   Fetched Rows: 7				
#	ID	NOME	CPF	TELEFONE
1	2	Cliente 5	444.444.444-44	(44) 4444-4444
2	3	Clie	444.444.444-44	(44) 4444-4444
3	5	Rod	322.223.455-43	95966-1234
4	6	Ped	322.223.455-43	95966-1234
5	7	Pedro	322.223.455-43	95966-1234
6	8	Cliente 5	444.444.444-44	(44) 4444-4444
7	9	Cliente 10	444.444.444-44	(44) 4444-4444



# Adicionando Chave Estrangeira

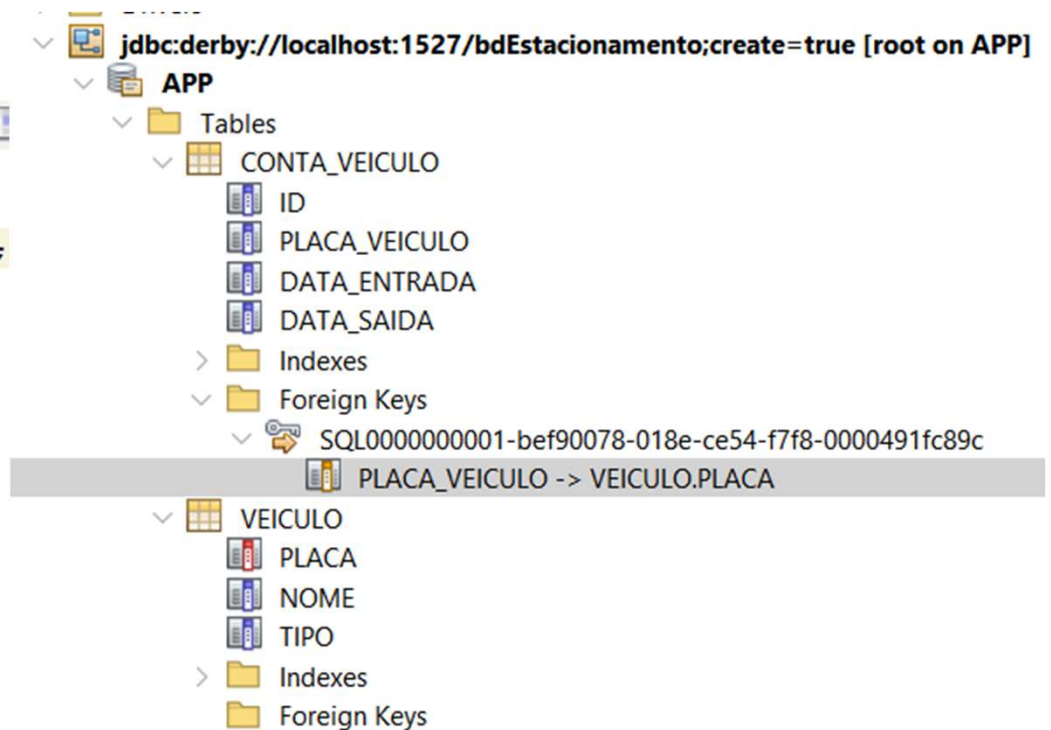
Após a criação das tabelas, adicione os relacionamentos, utilizando o código de exemplo.

Connection: jdbc:derby://localhost:1527/bdEstacionamento;create=true [root on ...]

```
1 Alter Table NOME_TABELA
2 Add FOREIGN KEY (NOME_CAMPO)
3 References NOME_TABELA_REFERENCIA (NOME_CAMPO_REFERENCIA);
```

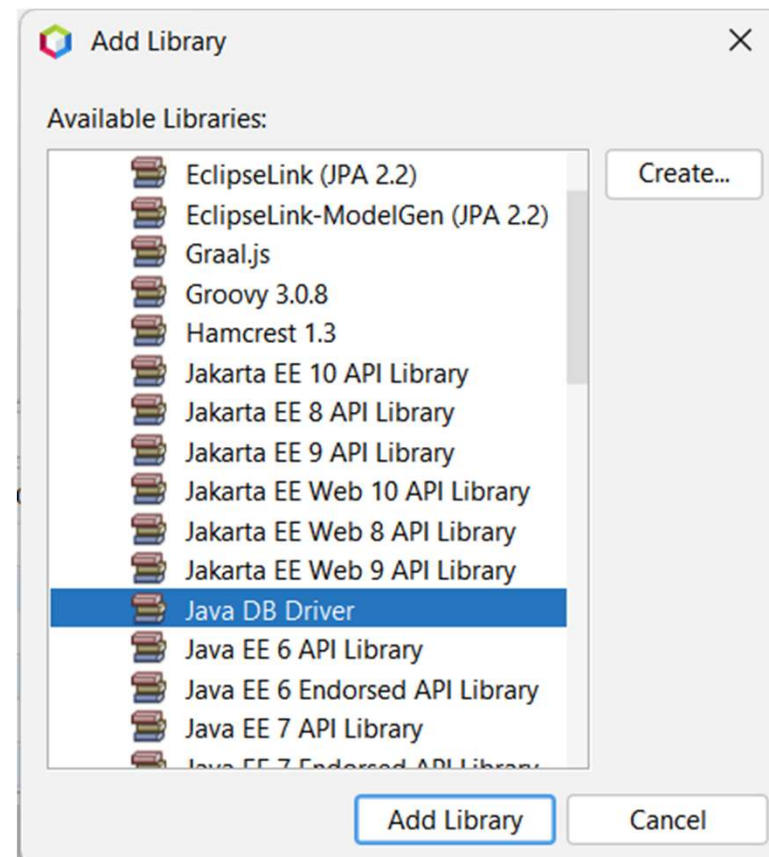
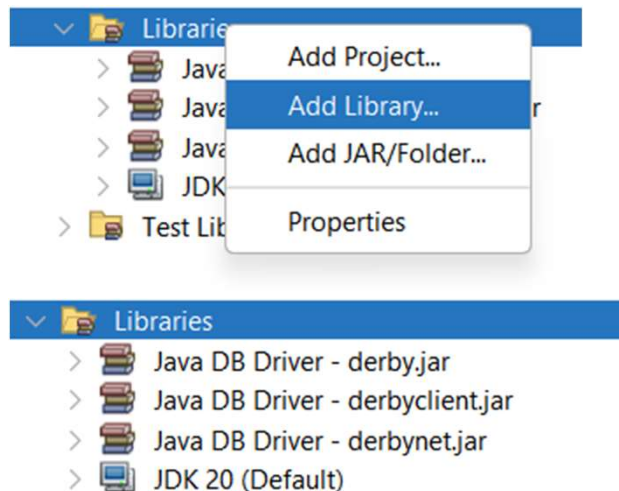
Exemplo:

```
1 Alter Table APP.CONTA_VEICULO
2 Add FOREIGN KEY (PLACA_VEICULO)
3 References APP.VEICULO (PLACA);
```



# Adicionando as Bibliotecas Java DB

- Na árvore do projeto, adicione a biblioteca **JavaDB**, para permitir a conexão ao banco de dados.



# Classes de Teste para conexão ao BD

- Inserir novos dados a tabela.

```
public class MainInsert {  
    public static void main(String[] args) throws SQLException, ClassNotFoundException {  
        Class.forName(className:"org.apache.derby.jdbc.ClientDriver");  
  
        String DATABASE_URL = "jdbc:derby://localhost:1527/bdteste";  
        String usuario = "APP";  
        String senha = "123";  
  
        Connection connection = DriverManager.getConnection(url:DATABASE_URL, user: usuario, password: senha);  
  
        String sql = "INSERT INTO CLIENTES (nome, cpf, telefone) VALUES (?, ?, ?)";  
        PreparedStatement ps = connection.prepareStatement(sql);  
  
        Cliente cliente = new Cliente(nome: "Cliente 10", cpf: "444.444.444-44", telefone: "(44) 4444-4444");  
  
        ps.setString(parameterIndex: 1, x: cliente.getNome());  
        ps.setString(parameterIndex: 2, x: cliente.getCpf());  
        ps.setString(parameterIndex: 3, x: cliente.getTelefone());  
        ps.execute();  
    }  
}
```

# Classes de Teste para conexão ao BD

- Atualizando os dados da tabela.

```
public class MainUpdate {  
    public static void main(String[] args) throws SQLException, ClassNotFoundException {  
        Class.forName(className:"org.apache.derby.jdbc.ClientDriver");  
  
        String DATABASE_URL = "jdbc:derby://localhost:1527/bdteste";  
        String usuario = "APP";  
        String senha = "123";  
  
        Connection connection = DriverManager.getConnection(url:DATABASE_URL, user:usuario, password:senha);  
  
        String sql = "UPDATE CLIENTES SET nome=?, cpf=?, telefone=? WHERE ID=?";  
        PreparedStatement ps = connection.prepareStatement(sql);  
  
        Cliente cliente = new Cliente(id: 1, nome: "Cliente 0", cpf: "000.000.000-00", telefone: "(00) 0000-0000");  
  
        ps.setString(parameterIndex: 1, x: cliente.getNome());  
        ps.setString(parameterIndex: 2, x: cliente.getCpf());  
        ps.setString(parameterIndex: 3, x: cliente.getTelefone());  
        ps.setInt(parameterIndex: 4, x: cliente.getId());  
        ps.execute();  
    }  
}
```

# Classes de Teste para conexão ao BD

- Retornando os dados da tabela.

```
public class MainSelect {  
    public static void main(String[] args) throws SQLException, ClassNotFoundException {  
        Class.forName(className:"org.apache.derby.jdbc.ClientDriver");  
  
        String DATABASE_URL = "jdbc:derby://localhost:1527/bdteste";  
        String usuario = "APP";  
        String senha = "123";  
  
        Connection connection = DriverManager.getConnection(url:DATABASE_URL, user: usuario, password: senha);  
  
        String sql = "SELECT * FROM CLIENTES";  
        PreparedStatement ps = connection.prepareStatement(sql);  
        ResultSet rs = ps.executeQuery();  
  
        while(rs.next()){  
            Cliente cliente = new Cliente();  
            cliente.setId(id: rs.getInt(columnLabel: "id"));  
            cliente.setNome(nome: rs.getString(columnLabel: "nome"));  
            cliente.setCpf(cpf: rs.getString(columnLabel: "cpf"));  
            cliente.setTelefone(telefone: rs.getString(columnLabel: "telefone"));  
            System.out.println(x: cliente);  
        }  
    }  
}
```



# Classes de Teste para conexão ao BD

- Excluindo os dados da tabela.

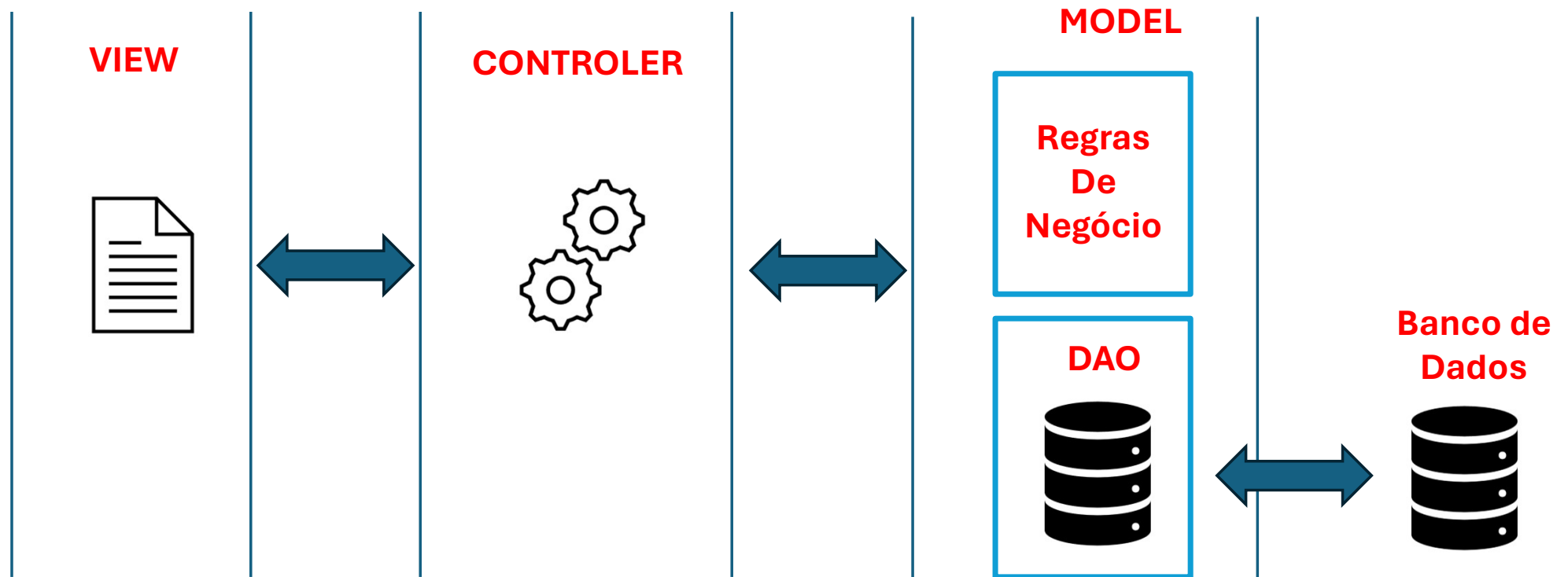
```
public class MainDelete {  
    public static void main(String[] args) throws SQLException, ClassNotFoundException {  
        Class.forName(className:"org.apache.derby.jdbc.ClientDriver");  
  
        String DATABASE_URL = "jdbc:derby://localhost:1527/bdteste";  
        String usuario = "APP";  
        String senha = "123";  
  
        Connection connection = DriverManager.getConnection(url:DATABASE_URL, user: usuario, password: senha);  
  
        String sql = "DELETE FROM CLIENTES WHERE ID=1";  
        PreparedStatement ps = connection.prepareStatement(sql);  
  
        ps.execute();  
    }  
}
```



# DAO (Data Access Object)

- O principal objetivo do padrão ***Data Access Object (DAO)*** é encapsular o acesso ao ***data source*** fornecendo uma interface para que as diversas outras camadas da aplicação possam se comunicar com o ***data source***.

# DAO (Data Access Object)





# Objeto ➔ Persistência dos Dados

## Modelo: Cliente.java

```
public class Cliente {  
    private String nome;  
    private String cpf;  
    private String telefone;  
  
    public String getNome() {  
        return nome;  
    }  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getCpf() {  
        return cpf;  
    }  
    public void setCpf(String cpf) {  
        this.cpf = cpf;  
    }  
  
    public String getTelefone() {  
        return telefone;  
    }  
    public void setTelefone(String telefone) {  
        this.telefone = telefone;  
    }  
}
```

## DAO: ClienteDAO.java

```
public class ClienteDAO {  
  
    private Connection connection;  
  
    public boolean inserir(Cliente cliente){  
        String sql = "INSERT INTO CLIENTES (nome, cpf, telefone) VALUES (?, ?, ?)";  
        ...  
    }  
  
    public boolean alterar(Cliente cliente){  
        String sql = "UPDATE CLIENTES SET nome=?, cpf=?, telefone=? WHERE ID=?";  
        ...  
    }  
  
    public boolean remover(Cliente cliente){  
        String sql = "DELETE FROM CLIENTES WHERE ID=1";  
        ...  
    }  
  
    public List<Cliente> listar(Cliente cliente){  
        String sql = "SELECT * FROM CLIENTES";  
        ...  
    }  
  
    public Cliente buscar(Cliente cliente){  
        String sql = "SELECT * FROM CLIENTES WHERE ID=?";  
        ...  
    }  
}
```

# ClienteDAO (Construtor)

```
public class ClienteDAO {  
  
    private Connection connection;  
  
    public ClienteDAO() {  
        try {  
            Class.forName("org.apache.derby.jdbc.ClientDriver");  
            String DATABASE_URL = "jdbc:derby://localhost:1527/bdteste";  
            String usuario = "root";  
            String senha = "123";  
            this.connection = DriverManager.getConnection(DATABASE_URL, usuario, senha);  
        } catch (ClassNotFoundException | SQLException ex) {  
            Logger.getLogger(ClienteDAO.class.getName()).log(Level.SEVERE, null, ex);  
        }  
    }  
}
```

# ClienteDAO (Select)

```
public class ClienteDAO {  
    private Connection connection;  
    public ClienteDAO() { ...11 linhas }  
  
    public List<Cliente> listar() {  
        String sql = "SELECT * FROM clientes";  
        List<Cliente> retorno = new ArrayList<>();  
        try {  
            PreparedStatement stmt = connection.prepareStatement(sql);  
            ResultSet resultado = stmt.executeQuery();  
            while (resultado.next()) {  
                Cliente cliente = new Cliente();  
                cliente.setId(resultado.getInt("id"));  
                cliente.setNome(resultado.getString("nome"));  
                cliente.setCpf(resultado.getString("cpf"));  
                cliente.setTelefone(resultado.getString("telefone"));  
                retorno.add(cliente);  
            }  
        } catch (SQLException ex) {  
            Logger.getLogger(ClienteDAO.class.getName()).log(Level.SEVERE, null, ex);  
        }  
        return retorno;  
    }  
}
```

# ClienteDAO (Insert)

```
public class ClienteDAO {  
  
    private Connection connection;  
  
    public ClienteDAO() { ...11 linhas }  
  
    public List<Cliente> listar() { ...19 linhas }  
  
    public boolean inserir(Cliente cliente) {  
        String sql = "INSERT INTO clientes(nome, cpf, telefone) VALUES(?,?,?)";  
        try {  
            PreparedStatement stmt = connection.prepareStatement(sql);  
            stmt.setString(1, cliente.getNome());  
            stmt.setString(2, cliente.getCpf());  
            stmt.setString(3, cliente.getTelefone());  
            stmt.execute();  
            return true;  
        } catch (SQLException ex) {  
            Logger.getLogger(ClienteDAO.class.getName()).log(Level.SEVERE, null, ex);  
            return false;  
        }  
    }  
}
```

# ClienteDAO (Update)

```
public class ClienteDAO {  
  
    private Connection connection;  
  
    public ClienteDAO() {...11 linhas }  
  
    public List<Cliente> listar() {...19 linhas }  
  
    public boolean inserir(Cliente cliente) {...14 linhas }  
  
    public boolean alterar(Cliente cliente) {  
        String sql = "UPDATE clientes SET nome=?, cpf=?, telefone=? WHERE id=?";  
        try {  
            PreparedStatement stmt = connection.prepareStatement(sql);  
            stmt.setString(1, cliente.getNome());  
            stmt.setString(2, cliente.getCpf());  
            stmt.setString(3, cliente.getTelefone());  
            stmt.setInt(4, cliente.getId());  
            stmt.execute();  
            return true;  
        } catch (SQLException ex) {  
            Logger.getLogger(ClienteDAO.class.getName()).log(Level.SEVERE, null, ex);  
            return false;  
        }  
    }  
}
```



# ClienteDAO (Delete)

```
public class ClienteDAO {  
  
    private Connection connection;  
  
    public ClienteDAO() { ...11 linhas }  
  
    public List<Cliente> listar() { ...19 linhas }  
  
    public boolean inserir(Cliente cliente) { ...14 linhas }  
  
    public boolean alterar(Cliente cliente) { ...15 linhas }  
  
    public boolean remover(Integer id) {  
        String sql = "DELETE FROM clientes WHERE id=?";  
        try {  
            PreparedStatement stmt = connection.prepareStatement(sql);  
            stmt.setInt(1, id);  
            stmt.execute();  
            return true;  
        } catch (SQLException ex) {  
            Logger.getLogger(ClienteDAO.class.getName()).log(Level.SEVERE, null, ex);  
            return false;  
        }  
    }  
}
```

# Exercício

- Crie um banco de dados Java DB para o aplicativo de estacionamento desenvolvido na última aula.
- Crie as classes de acesso ao banco de dados utilizando o padrão DAO.
- Modifique a classe de controle para salvar os dados da aplicação no BD, utilizando as classes DAO.