Lab 1 – Java DataBase Connectivity (JDBC)

Neste laboratório faremos o uso da **API JDBC** para conectar ao **Banco de dados PostgreSQL** e executar instruções **SQL** como: **CREATE, INSERT, SELECT, UPDATE** e **DELETE**.

Utilizando o projeto realizado no modulo orientação a objetos, iremos modificar para que o mesmo utilize banco de dados, realizando as mesmas funções que o mesmo realizava.

Exercícios

Exercício 1: Instalar o **PostgreSQL** e criar o **Banco de Dados threeway**.

Exercício 2: Criar todas as tabelas referente ao projeto realizado no modulo de orientação a objetos.

Exercício 3: Criar e Testar a classe **FabricaConexao.java**

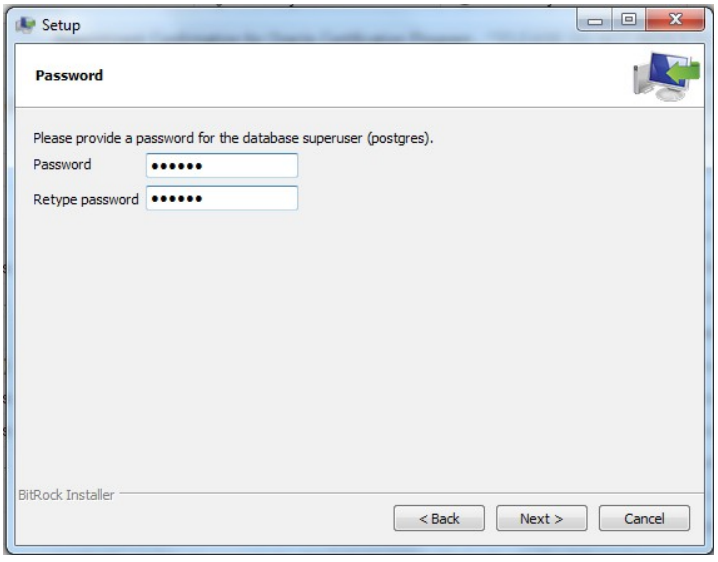
Exercício 4: Modificar a classe **ClienteDao.java** para acessar o banco de dados e fazer as operações de CRUD (Listagem, Insert, Update e Delete)

Exercício 5: Modificar toda a aplicação banco realizada nos laboratórios já realizados para que este esteja acessando banco de dados.

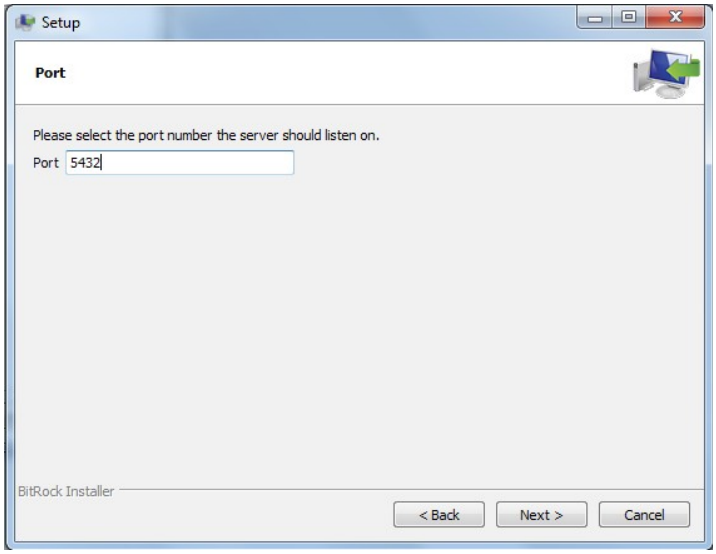
Exercício 1 – Instalar o PostgreSQL

1. Faça Download do instalador do postgreSQL no link: http://www.postgresql.org/download/.

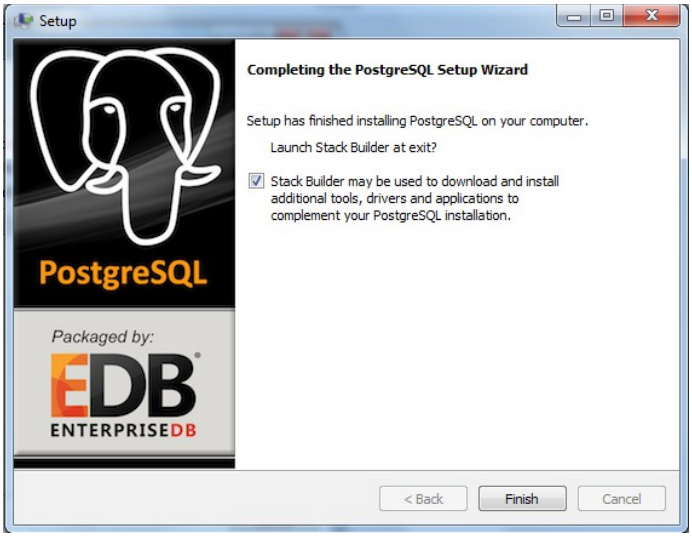
2. Execute o arquivo **postgresql-9.3.1-1-windows.msi**, selecione **Start** e depois selecione **NEXT** com as opções **Default** até a tela abaixo.



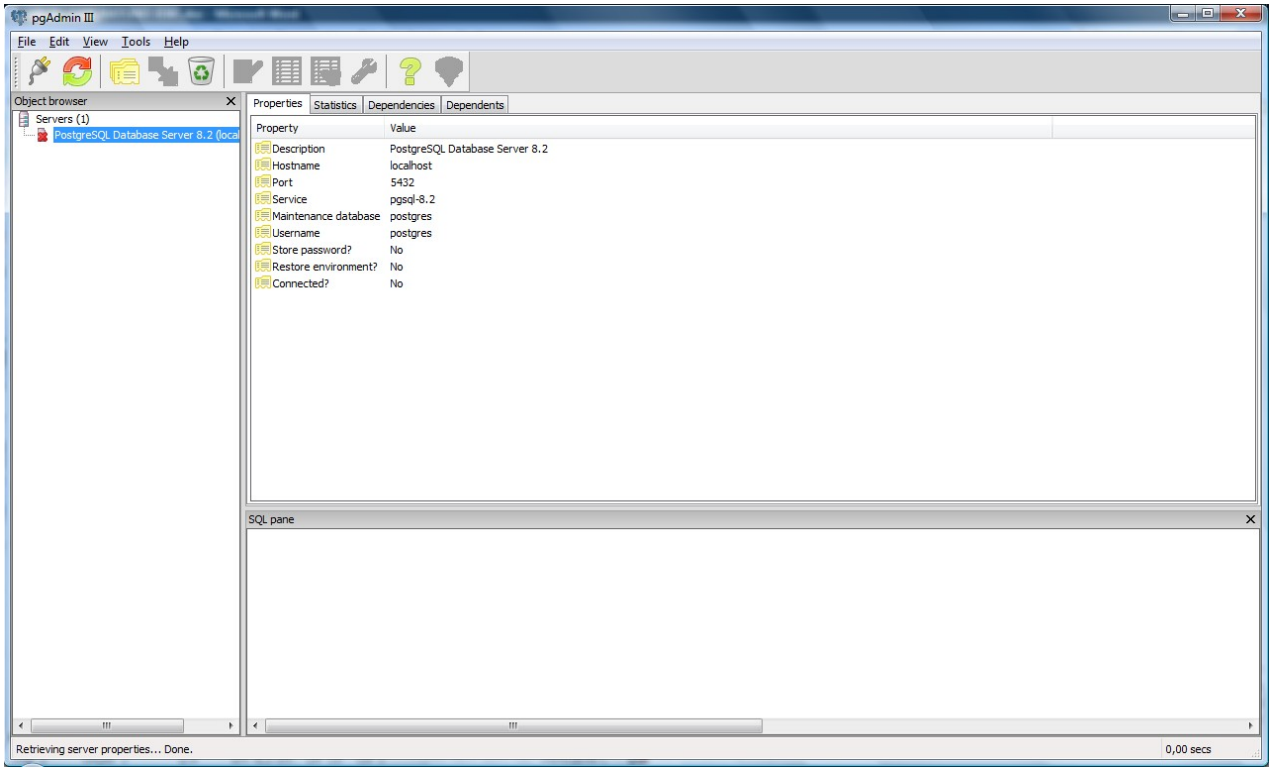
3. Informe a senha **123456**, para que todas as máquinas fiquem com a mesma senha de acesso e repita a senha **123456**, e selecione **Next**.



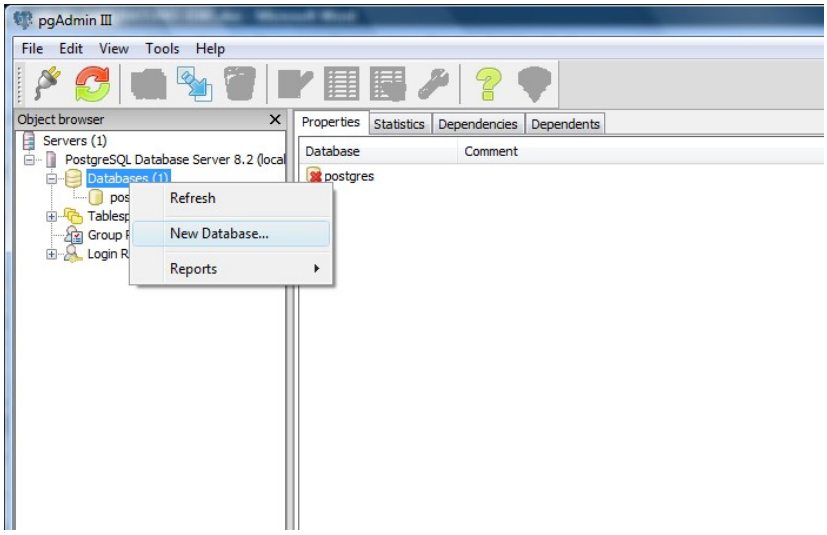
4. Confirme a porta do postgreSQL, a porta padrão e 5432

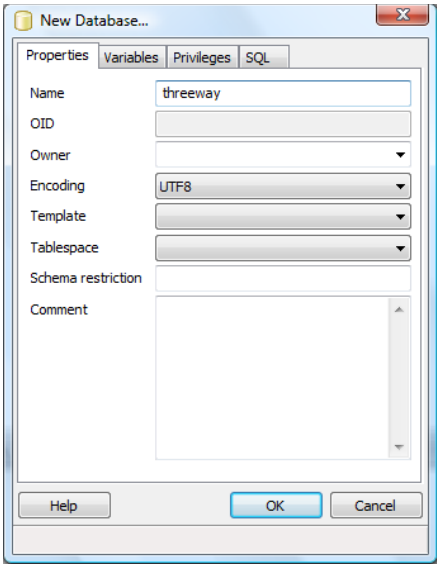


5. Após a instalação finalizar com sucesso, no menu **Iniciar,** selecione **pgAdmin III**. Esta é a ferramenta default de administração do banco de dados. Ao iniciá-la dê um duplo click na opção **PostgreSQL 9.3** para conectar ao banco de dados.



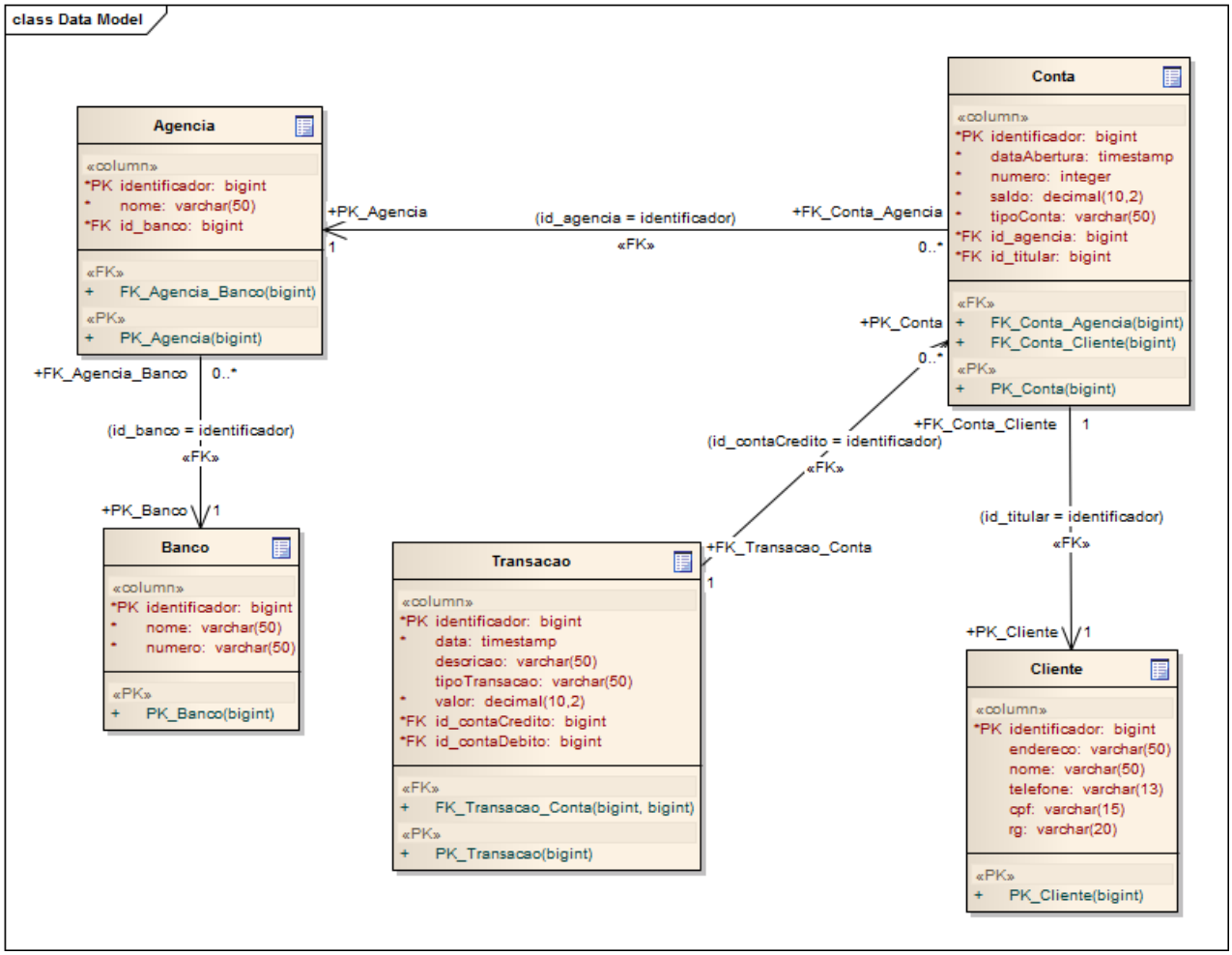
6. Clique com o botão direito do mouse em cima de **DataBases** e selecione a opção **NEW**, digite o nome **threeway**, selecione o **encoding UTF-8** e pressione **OK**. O banco será criado com toda estrutura necessária para criar as tabelas e acesso aos dados.





Exercício 2 – Criar todas as tabelas referente ao projeto realizado no modulo de orientação a objetos.

1. Crie o banco de dados de acordo com o MER abaixo.



2. O sql para a geração das tabelas conforme o MER acima:

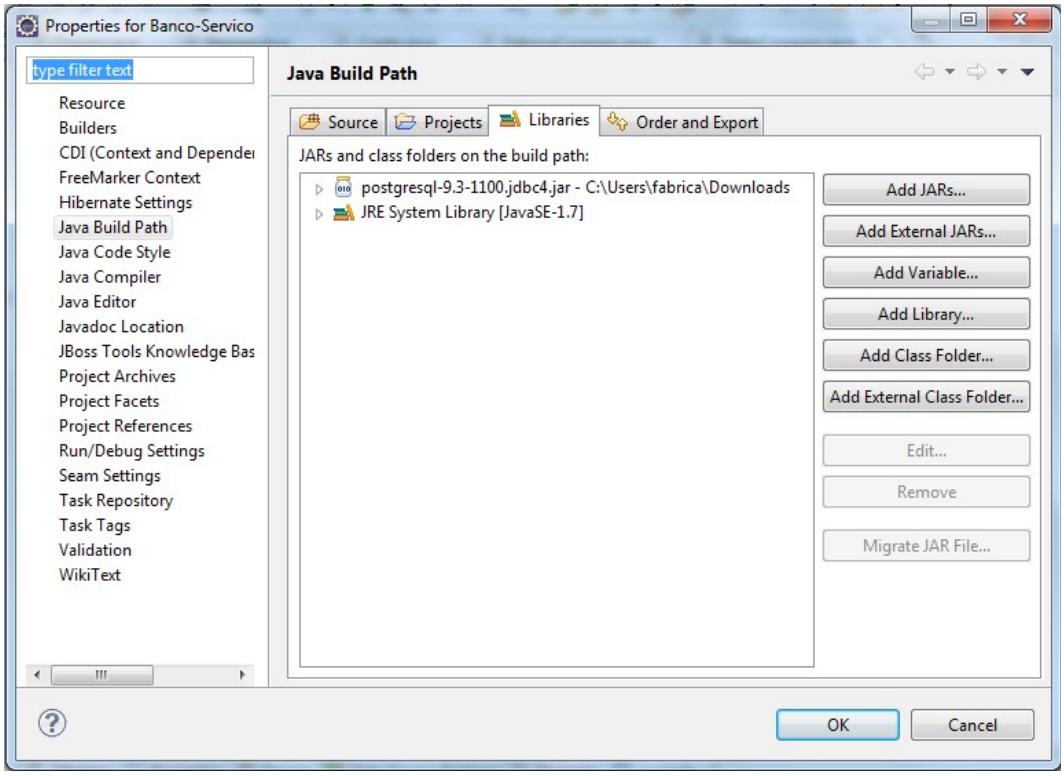
```
CREATE TABLE Banco (
        identificador serial NOT NULL,
        nome varchar(50) NOT NULL,
        numero varchar(50) NOT NULL
);
CREATE TABLE Agencia (
        identificador serial NOT NULL,
        nome varchar(50) NOT NULL,
        id_banco bigint NOT NULL
);
CREATE TABLE Conta (
        identificador serial NOT NULL,
        dataAbertura timestamp NOT NULL,
        numero integer NOT NULL,
        saldo decimal(10,2) NOT NULL,
        tipoConta varchar(50) NOT NULL,
        id_agencia bigint NOT NULL,
        id_titular bigint NOT NULL
);
```

```
CREATE TABLE Transacao (
        identificador serial NOT NULL,
        data timestamp NOT NULL,
        descricao varchar(50),
        tipoTransacao varchar(50),
        valor decimal(10,2) NOT NULL,
        id_contaCredito bigint,
        id_contaDebito bigint
);
CREATE TABLE Cliente (
        identificador serial NOT NULL,
        endereco varchar(50),
        nome varchar(50),
        telefone varchar(13),
        cpf varchar(15) NOT NULL,
        rg varchar(20)
);
ALTER TABLE Banco ADD CONSTRAINT PK_Banco
        PRIMARY KEY (identificador);

ALTER TABLE Agencia ADD CONSTRAINT PK_Agencia
        PRIMARY KEY (identificador);

ALTER TABLE Conta ADD CONSTRAINT PK_Conta
        PRIMARY KEY (identificador);

ALTER TABLE Transacao ADD CONSTRAINT PK_Transacao
        PRIMARY KEY (identificador);

ALTER TABLE Cliente ADD CONSTRAINT PK_Cliente
        PRIMARY KEY (identificador);

ALTER TABLE Agencia ADD CONSTRAINT FK_Agencia_Banco
        FOREIGN KEY (id_banco) REFERENCES Banco (identificador);

ALTER TABLE Conta ADD CONSTRAINT FK_Conta_Agencia
        FOREIGN KEY (id_agencia) REFERENCES Agencia (identificador);

ALTER TABLE Conta ADD CONSTRAINT FK_Conta_Cliente
        FOREIGN KEY (id_titular) REFERENCES Cliente (identificador);

ALTER TABLE Transacao ADD CONSTRAINT FK_Transacao_ContaCredito
        FOREIGN KEY (id_contaCredito) REFERENCES Conta (identificador);

ALTER TABLE Transacao ADD CONSTRAINT FK_Transacao_ContaDebito
        FOREIGN KEY (id_contaDebito) REFERENCES Conta (identificador);
```

Exercício 3 – Criar e testar a classe FabricaConexao

1. Use o projeto *Banco-Servico* para criação e configuração da classe de acesso ao banco de dados.

2. Vamos importar a biblioteca do **Postgres** para o projeto. Isso é necessário para a aplicação reconhecer o **Driver** de conexão com o banco. Faça download do driver Postgres em http://jdbc.postgresql.org/download.html#current, selecione o projeto e click com o botão direito do mouse e selecione a opção **Properties**, selecione **Java Build Path** e depois a aba **Libraries**, selecione a opção **Add External Jar...** e procure o driver do postgres e click em **OK**. Veja que ao selecionar ele irá aparecer na aba **Libraries** junto com a **JRE** conforme abaixo.



3. Crie uma classe chamada **FabricaConexao.java** dentro do **pacote DAO** que tenha o método **estático getConexao()** que retorne uma conexão com o Banco de Dados.

```
import java.sql.*;

public class FabricaConexao {
	static String url = "jdbc:postgresql://localhost:5432/threeway";
	static String usuario = "postgres";
	static String senha = "123456";

	public static Connection getConexao() throws SQLException {

		try{
			Class.forName("org.postgresql.Driver");
			return DriverManager.getConnection(url,usuario,senha);

		}catch(ClassNotFoundException e){
			throw new SQLException(e.getMessage());
		}
```

```
	}
}
```

4. Crie um classe para testar a conexão com o banco de dados, dê o nome de **TestaConexao.java**.

```
import java.sql.*;

public class TestaConexao {

	public static void main(String[] args) {
		Connection con;
		try {
			con = FabricaConexao.getConexao();
			if(con!=null)
				System.out.println("Conexao estabelecida!");

			con.close();

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

5. Execute a classe **TestaConexao.java**, se ela imprimir no console **Conexao estabelecida!** siga em frente.

Exercício 4 – Modificar a classe ClienteDao.java para acessar o banco de dados e fazer as operações de CRUD (Listagem, Insert, Update e Delete)

1. Modifique a classe **ClienteDao.java** criando de forma que seu construtor faça uma conexão com o banco de dados.

```
Connection conexao;

public ClienteDao(){

	try {
		conexao = FabricaConexao.getConexao();

	} catch (SQLException e) {

		System.out.println("Erro ao conectar com banco de dados" +
	e.getMessage());
	}
}
```

2. Vamos modificar o método que vai pegar as informações do objeto criado e inserir no banco de dados para persistir as informações e poderem ser recuperadas posteriormente. Modifique o método **salvar()** em **ClienteDao.java**.

```
@Override
public void salvar(Cliente entidade) {

        //Estamos criando a string de SQL que irá salvar o cliente.
        StringBuffer sql = new StringBuffer();

        sql.append("INSERT INTO cliente");

        sql.append("(endereco, nome, telefone, cpf, rg) VALUES (?,?,?,?,?)");

        try {

                PreparedStatement consulta = conexao.prepareStatement(sql.toString());

                consulta.setString(1, entidade.getEndereco());

                consulta.setString(2, entidade.getNome());

                consulta.setString(3, entidade.getTelefone());

                consulta.setString(4, entidade.getCpf());

                consulta.setString(5, entidade.getRg());

                consulta.executeUpdate();

        } catch (SQLException e) {

                System.out.println("Erro ao realizar Insert: " + e.getMessage());
        }

}
```

3. Agora modifique o método para alterar os dados do banco conforme os valores que estão preenchidos no objeto. Modifique o método **alterar()** em **ClienteDao.java**.

```
@Override
public void alterar(Cliente entidade) {

        //Estamos criando a string de SQL que irá alterar o cliente.
        StringBuilder sql = new StringBuilder();

        sql.append("UPDATE cliente SET ");

        sql.append("endereco = ?, ");

        sql.append("nome = ?, ");

        sql.append("telefone = ?, ");

        sql.append("cpf = ?, ");

        sql.append("rg = ? ");

        sql.append("WHERE identificador = ?");

        try {

                PreparedStatement consulta = conexao.prepareStatement(sql.toString());

                consulta.setString(1, entidade.getEndereco());

                consulta.setString(2, entidade.getNome());
```

```
                consulta.setString(3, entidade.getTelefone());

                consulta.setString(4, entidade.getCpf());

                consulta.setString(5, entidade.getRg());

                consulta.setLong(6, entidade.getIdentificador());

                consulta.executeUpdate();

            } catch (SQLException e) {

                System.out.println("Erro ao realizar Update: " + e.getMessage());
            }

        }
```

4. Modifique o método **excluir()** em **ClienteDao.java** para poder remover um registro do banco de dados.

```
@Override
public void remover(Cliente entidade) {

    //Estamos criando a string de SQL que irá remover  o cliente.
    String sql = "DELETE FROM cliente WHERE identificador = ?";

    try {

        PreparedStatement consulta = conexao.prepareStatement(sql);

        consulta.setLong(1, entidade.getIdentificador());

        consulta.executeUpdate();

    } catch (SQLException e) {

        System.out.println("Erro ao realizar Delete: " + e.getMessage());
    }

}
```

5. Modifique o método **obter(Long)** que recebe um parâmetro **Identificador** para a pesquisa em **ClienteDao.java** para poder recuperar um registro que foi inserido no banco de dados. Veja que se não for encontrado nenhum registro, então é retornado um objeto nulo (null).

```
@Override
public Cliente obter(Serializable identificador) {

    Cliente cliente = null;
    //Estamos criando a string de SQL que irá recuperar os dados de um cliente.
    String sql = "SELECT * FROM cliente WHERE identificador = ?";

    try {

        PreparedStatement consulta = conexao.prepareStatement(sql);

        consulta.setLong(1, (Long) identificador);

        ResultSet resultado = consulta.executeQuery();

        if(resultado.next()){

            cliente = new Cliente();

            cliente.setIdentificador(resultado.getLong("identificador"));
```

```
                cliente.setEndereco(resultado.getString("endereco"));
                cliente.setNome(resultado.getString("nome"));
                cliente.setTelefone(resultado.getString("telefone"));
                cliente.setCpf(resultado.getString("cpf"));
                cliente.setRg(resultado.getString("rg"));
            }
        } catch (SQLException e) {
            System.out.println("Erro ao realizar Select: " + e.getMessage());
        }
        return cliente;
    }
```

Note que nos passos 2, 3, 4 e 5 estamos criamos a string SQL e depois substituimos os “?” pelos valores da entidade passada como parâmetro.

6. Modifique o método **listar()**. Neste método não será passado parâmetro, ele simplesmente irá trazer todos os clientes que existem no banco de Dados.

```
@Override
public Collection<Cliente> listar() {
        ArrayList<Cliente> lista = new ArrayList<Cliente>();
        String sql = "SELECT * FROM cliente";
        try {
            PreparedStatement consulta = conexao.prepareStatement(sql);
            ResultSet resultado = consulta.executeQuery();
            while (resultado.next()) {
                Cliente cliente = new Cliente();
                cliente.setIdentificador(resultado.getLong("identificador"));
                cliente.setEndereco(resultado.getString("endereco"));
                cliente.setNome(resultado.getString("nome"));
                cliente.setTelefone(resultado.getString("telefone"));
                cliente.setCpf(resultado.getString("cpf"));
                cliente.setRg(resultado.getString("rg"));
                lista.add(cliente);
            }
        } catch (SQLException e) {
            System.out.println("Erro ao realizar Select: " + e.getMessage());
        }
        return lista;
```

```
}
```

7. Agora teste a aplicação banco com o módulo de manter clientes para testar o CRUD. Se precisar faça as alterações necessárias.

***Exercício 5* – Modifique toda a aplicação banco feita nos laboratórios já realizados para que este esteja acessando banco de dados.**