ANHANGUERA – CAMPUS GUAJAJARAS
THIAGO CONEGUNDES MORAIS - 3510640001
SISTEMA DE INFORMAÇÃO

PORTFÓLIO – RELATÓRIO DE AULA PRÁTICA:
PROGRAMAÇÃO WEB II
PARTE II

BELO HORIZONTE - MINAS GERAIS
MARÇO/2026

THIAGO CONEGUNDES MORAIS - 3510640001

PORTFÓLIO – RELATÓRIO DE AULA PRÁTICA

PROGRAMAÇÃO WEB II

PARTE II

Trabalho acadêmico apresentado ao Curso de Bacharelado em Sistema de Informação da Faculdade Anhanguera como requisito para obtenção de pontuação semestral.

Mediador Pedagógico: Rafael R. Fassula

BELO HORIZONTE - MINAS GERAIS
MARÇO/2026

SUMÁRIO

INTRODUÇÃO

O conceito de ORM (Object-Relational Mapping) é fundamental para o entendimento desse projeto. De acordo com a Dev.Media (2026), ORM é uma técnica que faz a “ponte” entre o mundo orientado a objetos que contempla as classes, objetos, herança, encapsulamento, conectando ao mundo relacional que contém tabelas, linhas, colunas, chaves primárias/estrangeiras.

Sem ORM é necessário escrever muitos scripts SQL, converter resultados de tabelas em objetos Java e controlar conexões, transações e mapeamentos, todos esses processos são executados manualmente, com ORM, isso é automatizado.

Hibernate é um framework de persistência de dados que atua como intermediário entre a aplicação Java e o banco de dados. A persistência de dados é a capacidade de guardar informações de forma permanente para que possam ser acessadas e utilizadas posteriormente e o Hibernate atua nesse processo. (DEV.MEDIA, 2026).

OBJETIVO

1) Construir uma aplicação Web utilizando as seguintes tecnologias:

- Linguagem de programação Java;
- Framework JSF(Java Server Faces);
- Framework Hibernate;
- IDE Eclipse; Servidor WildFlay;
- Banco de dados Oracle MySQL;

2) Construir um sistema Web Agenda de contatos com o objetivo de realizar cadastros enviados os dados para o banco de dados realizando manobras como inserção e busca de dados.

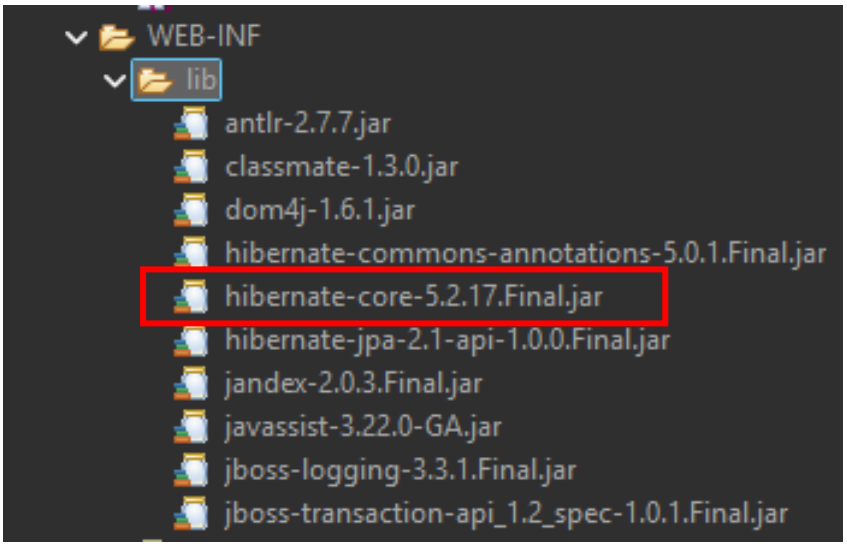
DESENVOLVIMENTO

Para iniciar o trabalho e necessário verificar se todas as versões estão convergentes, para isso temos:

- Compilador Java na versão 1.8.0_482;
- JDK na versão 1.8.0_482;
- Wild Fly na versão 15;
- Hibernate na versão 5.2.17 (Versão compatível com os tópicos acima);

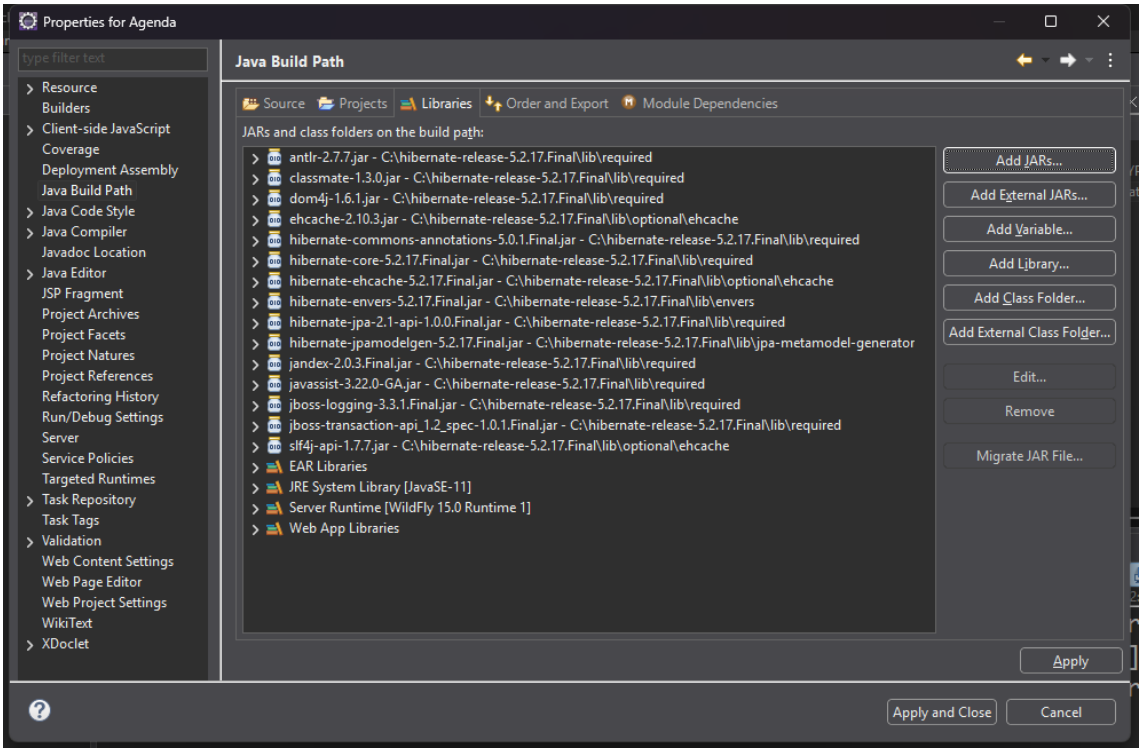
1) CONFIGURAÇÃO DO HIBERNATE

1.1) INSERÇÃO DOS ARQUIVOS E DEPENDÊNCIAS DO HIBERNATE

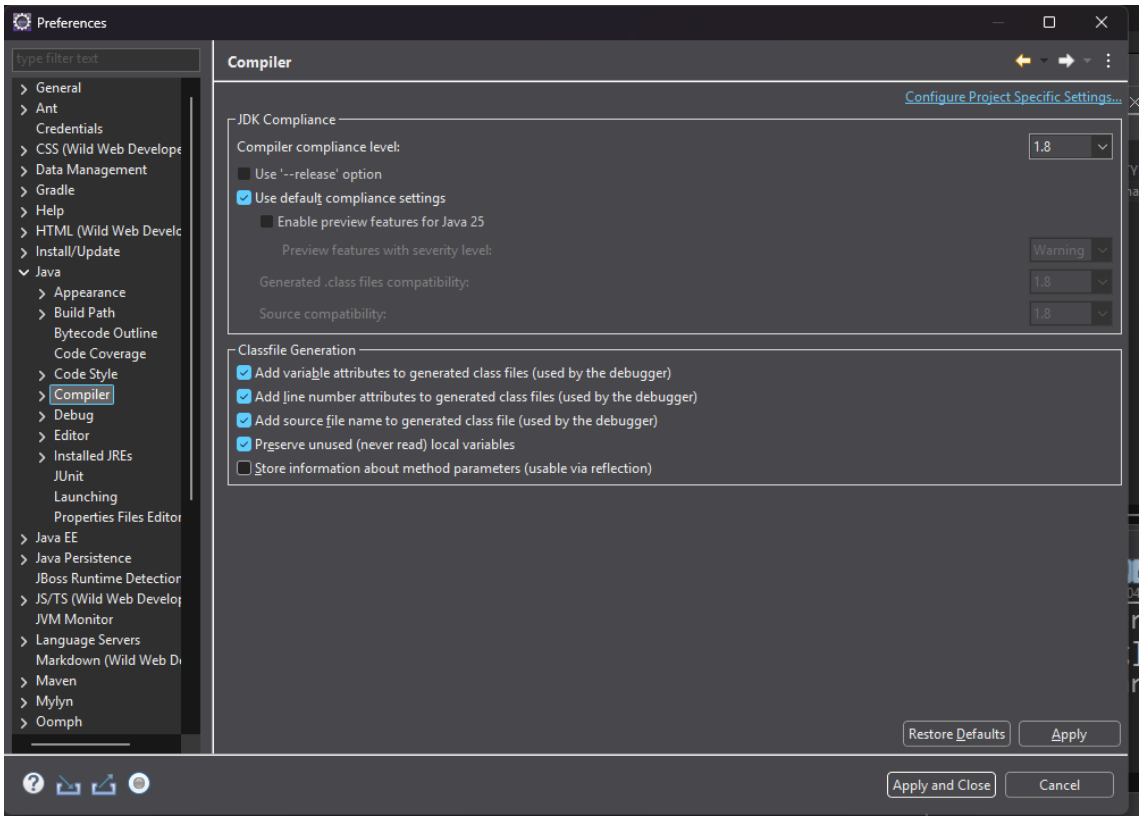


Arquivos .jar (Java Archive) do Hibernate são bibliotecas comprimidas que contêm o código pré-compilado (classes Java), e recursos necessários para integrar o framework Hibernate em um projeto Java. O arquivo hibernate-core.jar atua como núcleo funcional do Hibernate.

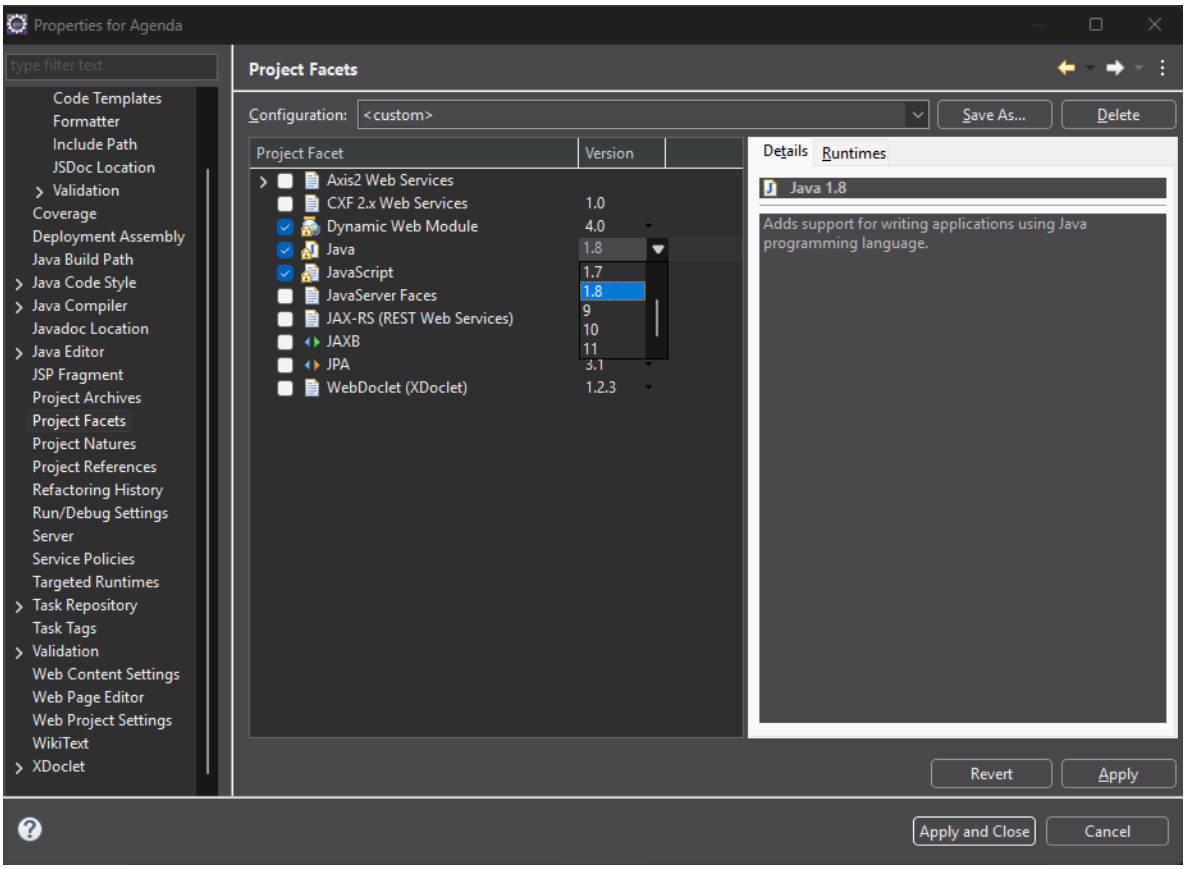
1.2) INSERÇÃO DAS JARS NO JAVA BUILD PATH PASSO 6



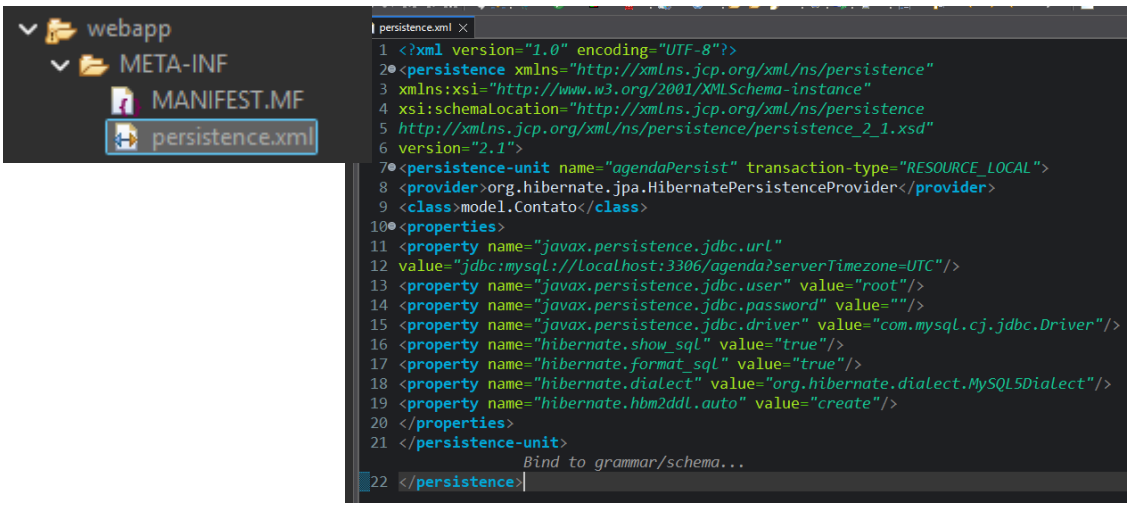
1.3) CONFIGURAÇÃO DO COMPILADOR PASSO 7



1.4) CONFIGURAÇÃO DO PROJECT FACETS PASSO 8D



1.5) CONFIGURAÇÃO DO PERSISTENCE.XML PASSO 8.5



2) ESTRUTURA DE PASTAS DO PROJETO AGENDA

3) APRESENTAÇÃO DO CÓDIGO FONTE - CONTATOMB.JAVA

ContatoMB.java

```java
package controller;

import javax.inject.Named;

@Named
@RequestScoped
public class ContatoMB {

    private Contato c = new Contato();

    private List<Contato> listaContatos;


    public ContatoMB() {
    }


    public String getId() {
        return String.valueOf(c.getId());
    }

    public void setId(String id) {
        c.setId(Integer.parseInt(id));
    }

    public String getNome() {
        return c.getNome();
    }

    public void setNome(String nome) {
        c.setNome(nome);
    }

    public String getSobrenome() {
        return c.getSobrenome();
    }

```

```
    public void setSobrenome(String sobrenome) {
        c.setSobrenome(sobrenome);
    }

    public String getTelefone() {
        return c.getTelefone();
    }

    public void setTelefone(String telefone) {
        c.setTelefone(telefone);
    }

    public String getEmail() {
        return c.getEmail();
    }

```

```
    public void setEmail(String email) {
        c.setEmail(email);
    }

    public void salvar() {
        try {
            System.out.println("=== TENTANDO SALVAR ===");
            System.out.println("Nome: " + c.getNome());
            System.out.println("Sobrenome: " + c.getSobrenome());
            System.out.println("Telefone: " + c.getTelefone());
            System.out.println("Email: " + c.getEmail());

            CRUD.inserir(c);

            System.out.println("=== SALVO COM SUCESSO ===");

            // Limpar o formulário após salvar
            c = new Contato();
            // Recarregar a lista
            listaContatos = CRUD.listarTodos();
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }

```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.1

A classe “**ContatoMB.java”** está dentro do pacote controller, ela faz a conexão entre o que está no front end, “**arquivo xhtml**”, e o “esqueleto”, “**contato.java”**.

Essa classe tem um objeto chamado “C” que tem relação com a classe contato. Esse objeto aciona os métodos get e set da classe contato, como explicado na primeira parte desse trabalho.

Na linha 17 foi utilizado um Array List com o nome: “listaContatos” essa lista é do tipo “contato”, ou seja, ela armazena todos os atributos delcarados na classe contato.

Na linha 64, no método salvar, na linha 64 a 70, foi implementado o System.out.println para averiguar se os dados estão de fato sendo resgatados método get.

3.1) APRESENTAÇÃO DO CÓDIGO FONTE - CONTATO.JAVA

```
1  package model;
2
3  import javax.persistence.*;
4  @Entity
5  @Table(name = "Contato")
6
7  public class Contato {
8
9      @Id
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     @Column
12     private int id;
13     @Column
14     private String nome;
15     @Column
16     private String sobrenome;
17     @Column
18     private String telefone;
19     @Column
20     private String email;
21
22     public int getId() {
23         return id;
24     }
25
```

```
26      public void setId(int id) {
27          this.id = id;
28      }
29
30      public String getNome() {
31          return nome;
32      }
33
34      public void setNome(String nome) {
35          this.nome = nome;
36      }
37
38      public String getSobrenome() {
39          return sobrenome;
40      }
41
```

```
42      public void setSobrenome(String sobrenome) {
43          this.sobrenome = sobrenome;
44      }
45
46      public String getTelefone() {
47          return telefone;
48      }
49
50      public void setTelefone(String telefone) {
51          this.telefone = telefone;
52      }
53
54      public String getEmail() {
55          return email;
56      }
57
58      public void setEmail(String email) {
59          this.email = email;
60      }
61
62  }
```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.1

A classe “**Contato.java**” funciona como arcabouço, esqueleto ela traz os atributos principais como nome telefone como variáveis privadas sendo necessário os métodos get e set para acessálas.

A anotação @Entity, na linha 4, marca a classe como entidade JPA (Java Persistence API) (API = Aplication Program Interface). Indica que os objetos dessa classe serão persistidos no banco de dados.

A anotoção @Table, linha 5, especifica qual tabela do banco de dados os dados serão direcionados.

O “@Id” Marca o campo id como chave primária da tabela, Significa que este campo é o identificador único de cada registro

@GeneratedValue(strategy = GenerationType.IDENTITY) Define como o valor do ID será gerado automaticamente IDENTITY = Delega ao banco de dados a responsabilidade de gerar o ID sendo equivalente ao AUTO_INCREMENT do MySQL

3.2) APRESENTAÇÃO DO CÓDIGO DO ARQUIVO CRUD.JAVA

```java
package model;

import java.util.List;



public class CRUD {
        public static void inserir(Contato c1) {

            try {
                EntityManager entityManager = DAO.getEntityManager();

                entityManager.getTransaction().begin();
                entityManager.persist(c1);
                entityManager.getTransaction().commit();
                entityManager.close();
                System.out.println("conectado Salvo!");
            } catch(Exception ex) {
                ex.printStackTrace();
            }
        }

```

```java
25          // NOVO MÉTODO PARA LISTAR TODOS
26          public static List<Contato> listarTodos() {
27              List<Contato> contatos = null;
28              try {
29                  EntityManager entityManager = DAO.getEntityManager();
30                  entityManager.getTransaction().begin();
31
32                  // Query JPQL para buscar todos os contatos
33                  contatos = entityManager.createQuery("FROM Contato", Contato.class).getResultList();
34
35                  entityManager.getTransaction().commit();
36                  entityManager.close();
37              } catch(Exception ex) {
38                  ex.printStackTrace();
39              }
40              return contatos;
41          }
42
```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.2

O arquivo **crud.java** tem a função de enviar dados para o banco, realizando os processos de inserção - “creat”; leitura - “read”; alteração “update”; deleção –“delete”.

Na linha 10 temos o método inserir, ele recebe como parâmetro o objeto C1 do tipo Contato. O **EntityManager** é o objeto que gerencia as operações com banco, já o **DAO.getEntityManager()** é método que cria/devolve uma conexão com o banco

O método **getTransaction()** acessa o controle de transação. O método **begin(),** Inicia uma transação e o método **persist(c1**) manda salvar o objeto c1 no banco.

Na linha 26, temos o método listarTodos(), na linha 33 o arrayList **contatos** que chama o método “entityManager.createQuery” criando uma query que traz todos os contatos contidos na tabela contatos no banco de dados.

3.3) APRESENTAÇÃO DO CÓDIGO DO ARQUIVO DAO.JAVA

DAO.java

```java
package model;

import javax.persistence.*;

public class DAO {

    private static final EntityManagerFactory emFactory;

    static {
        emFactory = Persistence.createEntityManagerFactory("agendaPersist");
    }

    public static EntityManager getEntityManager() {
        return emFactory.createEntityManager();
    }

    public static void fecharFactory() {
        emFactory.close();
    }

}
```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.3

O arquivo “DAO” significa Data Access Object, ele gerencia a fábrica de conexões com o banco de dados usando JPA (Java Persistence Aplication).

Na linha 7, temos o EntityManagerFactory emFactoy que representa Fábrica que cria conexões com o banco.

Na linha 10, o método “Persistence.createEntityManagerFactory()” cria a fábrica, já o parâmetro "agendaPersist" representa o nome da unidade de persistência definida no arquivo persistence.xml.

Na linha 13, o emFactory pede para criar uma nova entidade de conexão com o banco de dados.

3.4) APRESENTAÇÃO DO CÓDIGO DO ARQUIVO PERSISTENCE.XML

persistence.xml

```xml
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd (xsi:schemaLocation with catalog)
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
<persistence-unit name="agendaPersist" transaction-type="RESOURCE_LOCAL">
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
<class>model.Contato</class>
```

```xml
persistence.xml
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd (xsi:schemaLocation with catalog)
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
<persistence-unit name="agendaPersist" transaction-type="RESOURCE_LOCAL">
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
<class>model.Contato</class>
<properties>
<property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/agenda?serverTimezone=UTC"/>
<property name="javax.persistence.jdbc.user" value="root"/>
<property name="javax.persistence.jdbc.password" value="150589"/>
<property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
<property name="hibernate.show_sql" value="true"/>
<property name="hibernate.format_sql" value="true"/>
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>
<property name="hibernate.hbm2ddl.auto" value="update"/>
</properties>
</persistence-unit>
</persistence>
```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.4

O arquivo persistence.xml é o arquivo mais importante do JPA, nele é dito com se concectar com o banco de dados especificando: o banco, senha, propriedades da conexão, porta, etc...

```xml
13 <property name="javax.persistence.jdbc.user" value="root"/>
14 <property name="javax.persistence.jdbc.password" value="150589"/>
15 <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
16 <property name="hibernate.show_sql" value="true"/>
```

Na linha 13 temos o value “**root**” que é o nome da instancia do banco de dados, como também, a senha de acesso na linha 14, o conector jdbc driver na linha 15.

Na linha 16 a propriedade: **name="hibernate.show_sql" value="true"** mostra o sql no console do eclipse, muito utilizado para entender o código no processo de inspeção e “TheBug”.

Na linha 18 na propriedade value="org.hibernate.dialect.MySQL5Dialect mostra o dialeto SQL usado no MySQL 5 usado no workbench.

3.5) APRESENTAÇÃO DO CÓDIGO DO ARQUIVO CONTATO.XHTML

contato.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
    <meta charset="UTF-8" />
    <title>Agenda Virtual</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" />
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"></script>
</h:head>
<h:body >
    <div class="container mt-4 border rounded" >
        <h1 class="mb-4">Agenda</h1>

        <h:form>

            <div class="mb-3">
                <label class="form-label ">Nome:</label>
                <h:inputText value="#{contatoMB.nome}" styleClass="form-control w-50" />
            </div>

            <div class="mb-3">
                <label class="form-label ">Sobrenome:</label>
                <h:inputText value="#{contatoMB.sobrenome}" styleClass="form-control w-50" />
            </div>

            <div class="mb-3">
                <label class="form-label ">Telefone:</label>
                <h:inputText value="#{contatoMB.telefone}" styleClass="form-control w-50" />
            </div>

            <div class="mb-3">
                <label class="form-label ">E-mail:</label>
                <h:inputText value="#{contatoMB.email}" styleClass="form-control w-50" />
            </div>


            <div class="mt-4">
                <h:commandButton value="Salvar" action="#{contatoMB.salvar}" styleClass="btn btn-primary" />
            </div>
            <div class="mt-4">
               <h:link outcome="listar" value="Listar Todos" styleClass="btn btn-primary" />
            </div>
        </h:form>

        <div class="mt-4 p-3 border rounded bg-light">
            <h5 class="text-primary">Contato Cadastrado:</h5>
            <p class="mb-1">Código Id: #{contatoMB.id}</p>
            <p class="mb-1">Nome: #{contatoMB.nome} #{contatoMB.sobrenome}</p>
            <p class="mb-1">Telefone: #{contatoMB.telefone}</p>
            <p class="mb-1">E-mail: #{contatoMB.email}</p>

        </div>
    </div>
</h:body>
</html>
```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.5

O arquivo CONTATO.XHTML é o arquivo front end sendo a interface visual de interação com o usuário. Para essa interface, ao invés de usar um arquivo estilo.css foi utilizado o boot strap para trazer um designer mais profissional de forma mais simples.

```
8     <meta name="viewport" content="width=device-width, initial-scale=1" />
9     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" />
10    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"></script>
```

Na linha 9 e 10 trazemos a conexão com o boot strap. O boot strap é uma framework com uma estrutura de código que facilita no processo de estilização da interface gráfica.

```
16        <h:form>
17
18            <div class="mb-3">
19                <label class="form-label ">Nome:</label>
20                <h:inputText value="#{contatoMB.nome}" styleClass="form-control w-50" />
21            </div>
22
```

Na linha 20 temos a tag **<h:inputText>** representado um componente JSF para campo de texto. O styleClass = “form-control w-50” significa que o a caixa de entrada terá 50% da largura da tela.

O valor digitado pelo usuário sera encaminhado para o arquivo contatoMB.nome que ativará o objeto C com os métodos get e set enviado os dados para o contato.java,

```
38
39            <div class="mt-4">
40                <h:commandButton value="Salvar" action="#{contatoMB.salvar}" styleClass="btn
41            </div>
42            <div class="mt-4">
43               <h:link outcome="listar" value="Listar Todos" styleClass="btn btn-primary" />
44            </div>
45        </h:form>
46
```

Na linha 40, ao clicar no botão salvar, o método salvar da classe contatoMB. é ativado.

Ao acionar o método salvar, os dados percorrem o arquivo contato.xhmtl, passando pelo contatoMB.java e por fim ele chega ao contato.java; porem, na classe contatoMB, quando o método salvar é ativado, é invocado também o método inserir da calsse CRUD, fazendo que todos os dados sejam armazenados no banco de dados.

```java
public void salvar() {
    try {
        System.out.println("=== TENTANDO SALVAR ===");
        System.out.println("Nome: " + c.getNome());
        System.out.println("Sobrenome: " + c.getSobrenome());
        System.out.println("Telefone: " + c.getTelefone());
        System.out.println("Email: " + c.getEmail());

        CRUD.inserir(c);
```

```java
public class CRUD {
    public static void inserir(Contato c1) {

        try {
            EntityManager entityManager = DAO.getEntityManager();

            entityManager.getTransaction().begin();
            entityManager.persist(c1);
            entityManager.getTransaction().commit();
            entityManager.close();
            System.out.println("conectado Salvo!");
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }

}
```

E assim ocorre o processo de inserção de dados, processamento e armazenamento.

3.5) APRESENTAÇÃO DO CÓDIGO LISTAR.XHTML

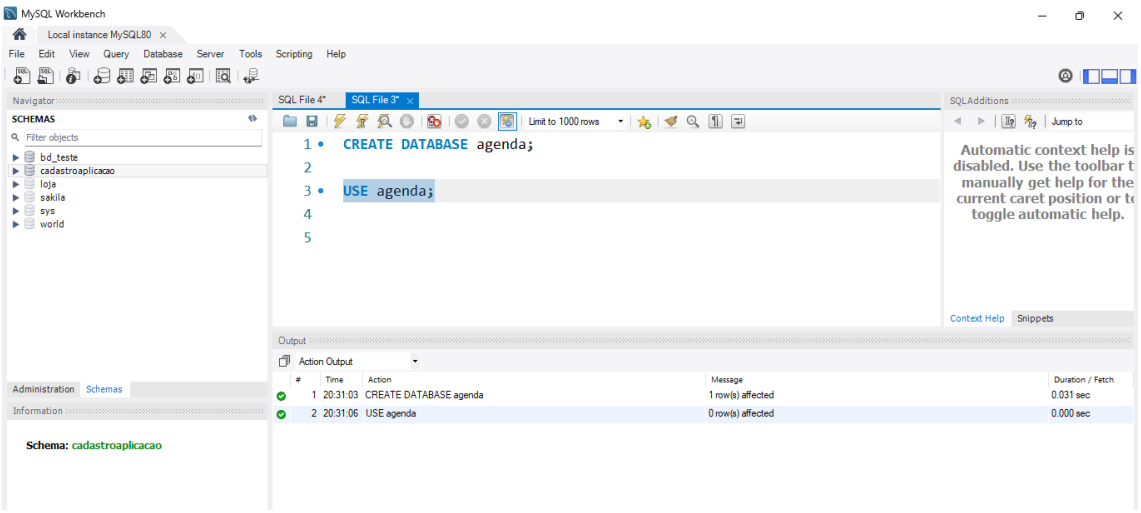
```xhtml
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head>
    <meta charset="UTF-8" />
    <title>Lista de Contatos</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" />
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"></script>
</h:head>
```

```xhtml
<h:body >
    <div class="container mt-4">
        <h1 class="mb-4">Lista de Contatos</h1>

        <h:link outcome="contato" value="Novo Contato" styleClass="btn btn-success mb-3" />

        <h:dataTable value="#{contatoMB.listaContatos}" var="contato"
                     styleClass="table table-striped table-bordered"
                     headerClass="table-dark">

            <h:column>
                <f:facet name="header">ID</f:facet>
                #{contato.id}
            </h:column>

            <h:column>
                <f:facet name="header">Nome</f:facet>
                #{contato.nome} #{contato.sobrenome}
            </h:column>
```

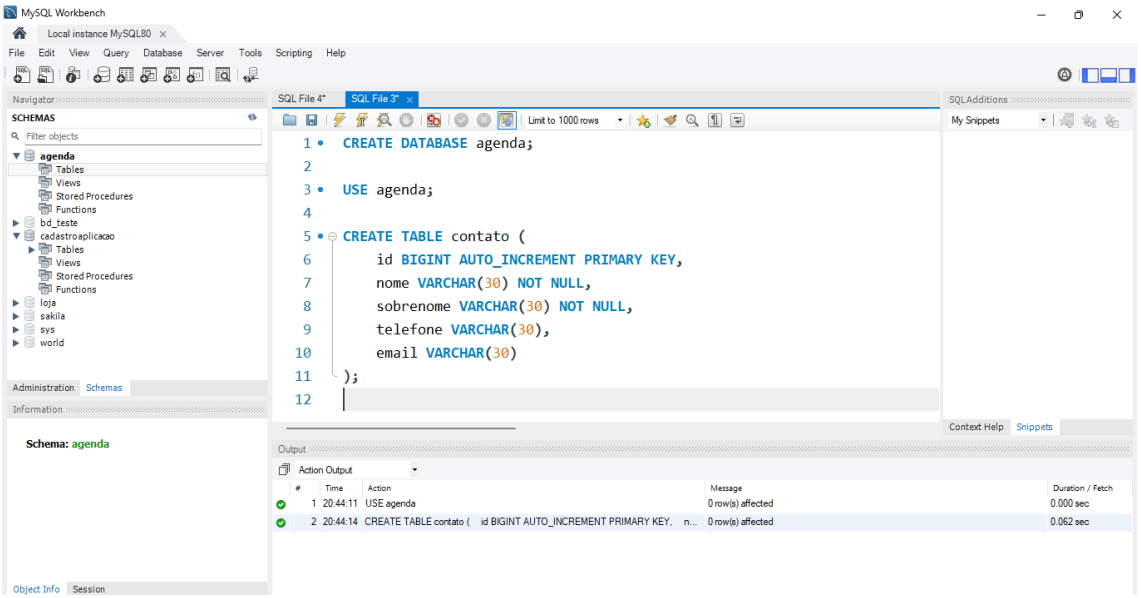
```xhtml
            <h:column>
                <f:facet name="header">Telefone</f:facet>
                #{contato.telefone}
            </h:column>

            <h:column>
                <f:facet name="header">Email</f:facet>
                #{contato.email}
            </h:column>

        </h:dataTable>

        <h:link outcome="contato" value="Voltar" styleClass="btn btn-secondary" />
    </div>
</h:body>
</html>
```

O arquivo **listar.xhtml** apresenta na tela de resultados que foram cadastrados e estão contidos no banco de dados.

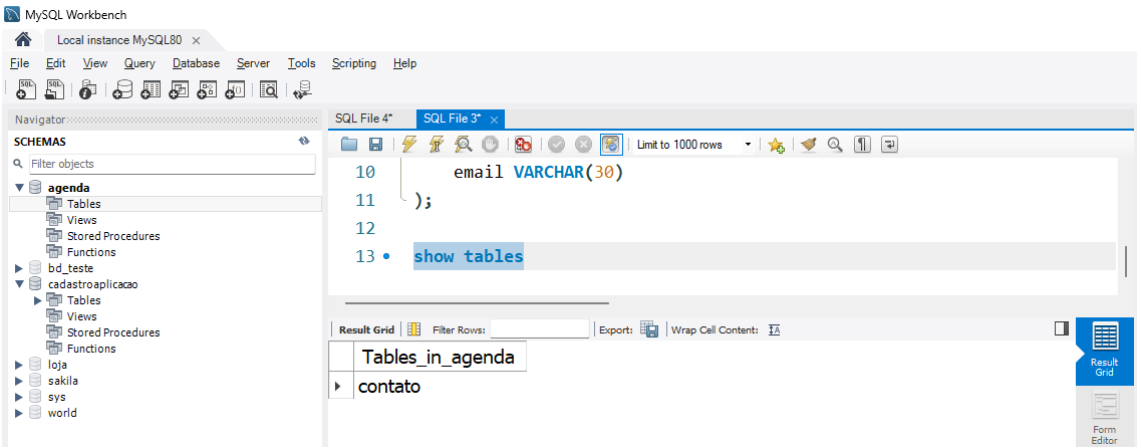
3.5) APRESENTAÇÃO DO BANCO DE DADOS



Criação do banco de dados agenda, criação da coluna contato e seus respectivos atributos.

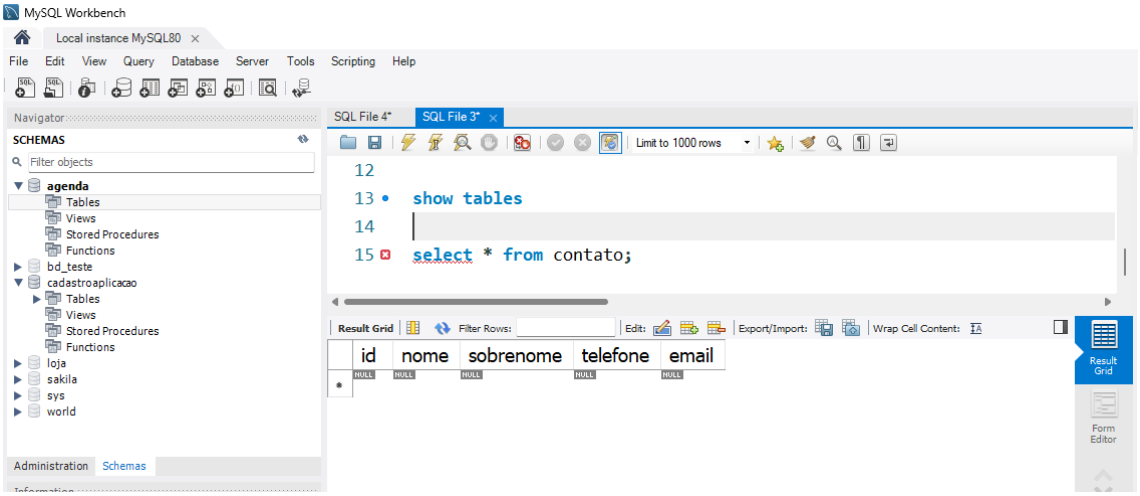


RESULTADO DA CRIAÇÃO DAS TABELAS

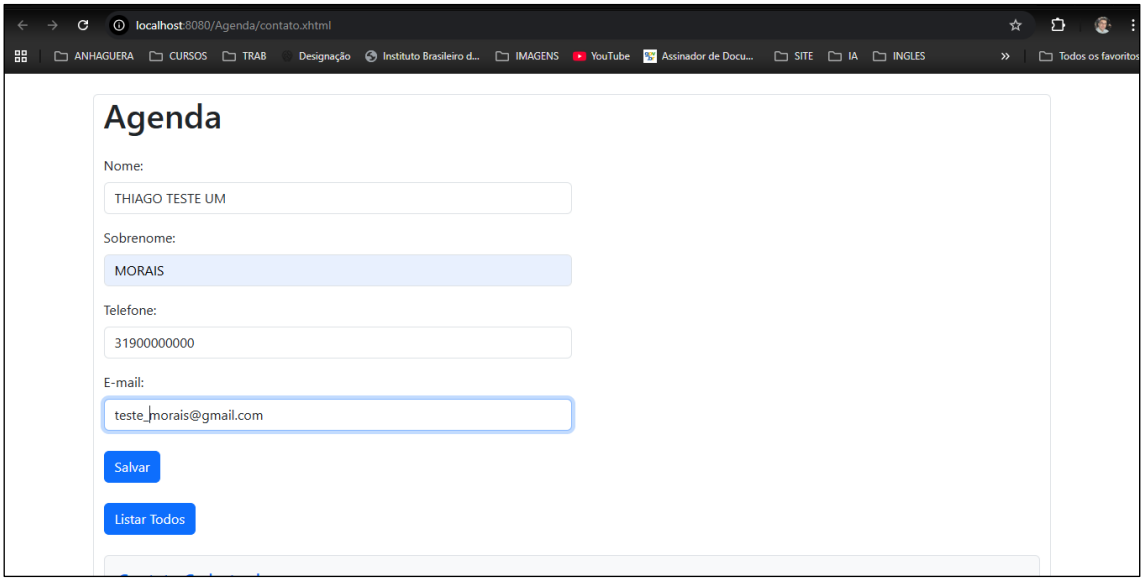


RESULTADO DO SELECT NA TABELA

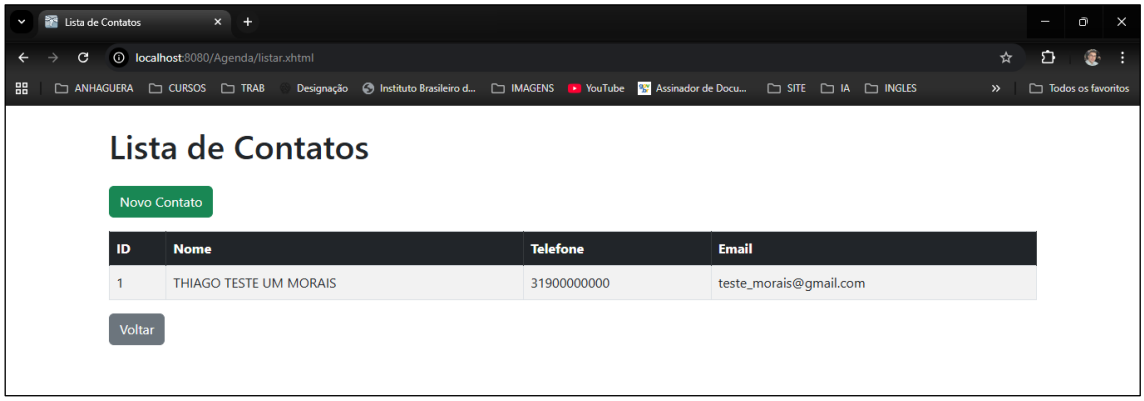
Observe que o banco de dados está vazio nesse momento.



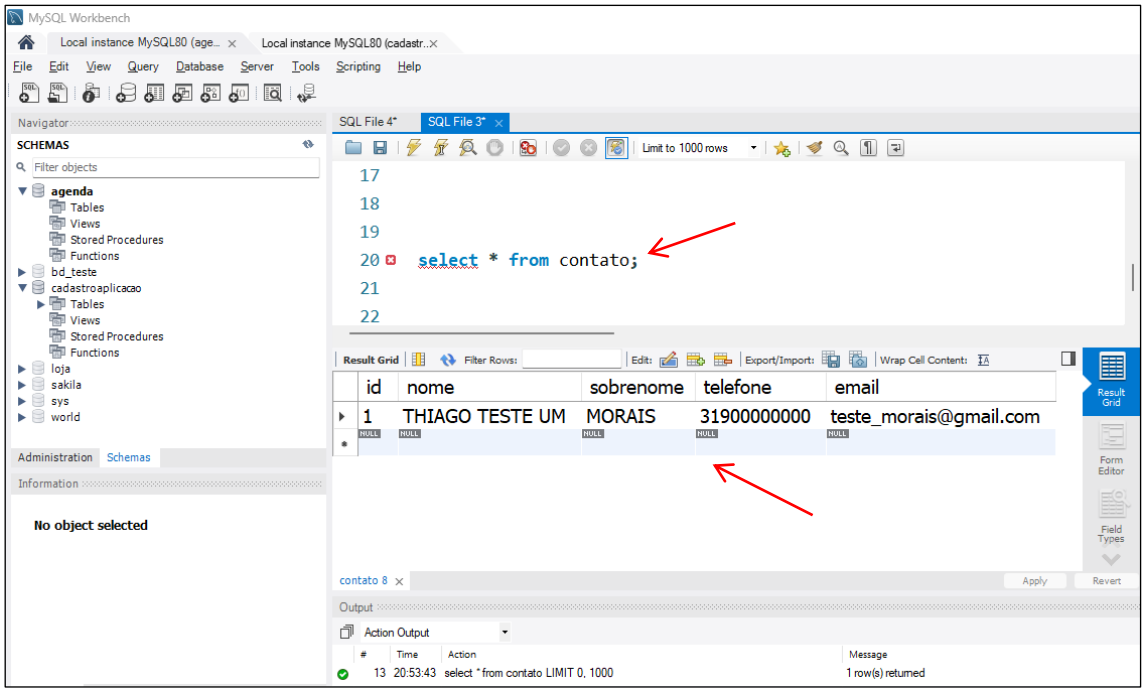
RESULTADO FINAL



Após “Salvar” e clicar em “Listar Todos”, temos:



Ao retornar ao banco de dados, temos:



CONSIDERAÇÕES FINAIS

Para executar esse trabalho foi possível superar muitos desafios passando pelos principais tópicos:

- Entendimento dos conceitos do Hibernate e ORM na prática;
- Junção de todos os conhecimentos: HTML, Boot Strap, JAVA Front End, Back End, Banco de dados.
- Assimilação dos conceitos do DAO e do arquivo persisntece.xml

Nessa jornada concluí que o hibernate facilita e sintetiza o processo de escrita de código otimizando o processo de desenvolvimento de sistema.

REFERÊNCIAS

1) DEV.MEDIA. Guia Completo de Hibernate: Aprenda Hibernate do Básico ao Avançado,2026 Disponível em m: https://www.devmedia.com.br/guia/hibernate. Acesso em 18 fev. 2026.

2) DEV.MEDIA. Guia completo de JSF (JavaServer Faces): aprenda JSF agora**!** Disponível em: *https://www.devmedia.com.br/guia/jsf-ja*vaserver. Acesso em: [Março 2026]

3) CONFIGURANDO O SERVIDOR WILDFLY COM A IDE ECLIPSE. Disponível em: https://www.youtube.com/watch?v=pKqQXSUypow. Acesso em: 13 fev. 2026.