Compte Rendu du TP Gestion des employés (MVC, DAO, Java Swing)

Réalisé par **Aya El Alama**

Le 07 Décembre 2024
Année Universitaire : 2024/2025

0.1 Introduction

Ce rapport présente les résultats du travail pratique portant sur la gestion des employés à travers une application Java utilisant les principes du modèle **MVC (Modèle-Vue-Contrôleur)** et l'architecture **DAO (Data Access Object)**. Le TP a pour objectif de développer une application de gestion d'employés avec Swing pour l'interface utilisateur en appliquant le modèle MVC.

0.1.1 Objectifs du TP

L'objectif principal de ce TP était de comprendre et d'implémenter le modèle **MVC** pour séparer les différentes couches de l'application et utiliser le modèle **DAO** pour gérer l'accès aux données. À la fin du TP, nous devions réaliser une application capable de :

- Gérer les employés dans un système de base de données.
- Permettre à l'utilisateur d'ajouter, modifier, ou supprimer des informations sur les employés.
- Afficher ces informations dans une interface graphique avec Java Swing.

0.1.2 Méthodologie

Pour atteindre ces objectifs, nous avons utilisé plusieurs étapes :

1. **Conception du modèle de données** : Création de la classe 'Employe' et des différentes méthodes de gestion des données.
2. **Implémentation de la couche DAO** : Mise en place des méthodes pour accéder aux données via une base de données.
3. **Développement de l'interface utilisateur avec Java Swing** : Création de fenêtres pour interagir avec l'utilisateur.
4. **Séparation des responsabilités avec le modèle MVC** : Organisation de l'application en trois parties distinctes.

0.2 Structure des Classes dans Eclipse

L'application repose sur une architecture claire et organisée, respectant le modèle MVC (Modèle-Vue-Contrôleur). La capture d'écran ci-dessous montre la structure des fichiers telle qu'elle apparaît dans Eclipse :

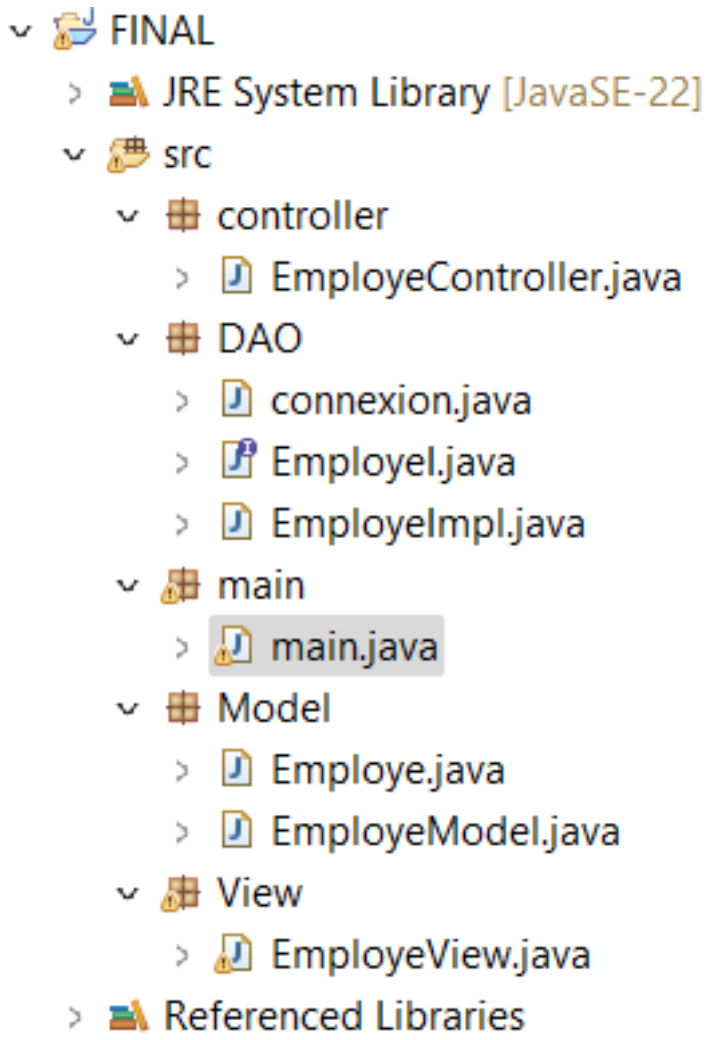

Figure 1: Structure des classes dans Eclipse

Comme on peut le voir dans la capture ci-dessus:

- **Dossier** `src` : Contient le code source de l'application.
- **Package** `model` : Définit les classes de données, comme `Employe`.

- **Package** `dao` : Contient les classes pour gérer l'interaction avec la base de données, comme `EmployeIDAO`.
- **Package** `view` : Regroupe les classes qui gèrent l'interface utilisateur via Java Swing.
- **Fichier** `Main.java` : Le point d'entrée de l'application, qui initialise et lie les différentes composantes.

0.3 Base de Données

Pour la gestion des données, j'ai utilisé une base de données locale hébergée sur **localhost** avec l'outil **phpMyAdmin**. Cet environnement permet de manipuler facilement les tables et d'exécuter des requêtes SQL.

La table principale, nommée `employee`, est utilisée pour stocker les informations des employés. Cette table contient les colonnes suivantes:

- **id** : Identifiant unique de l'employé (type entier, clé primaire).
- **nom** : Nom de l'employé (type chaîne de caractères).
- **prenom** : Prénom de l'employé (type chaîne de caractères).
- **poste** : Poste occupé par l'employé (type chaîne de caractères).
- **date_embauche** : Date d'embauche (type date).
- **email** : Adresse email de l'employé (type chaîne de caractères).
- **salaire** : Salaire mensuel de l'employé (type décimal).

L'image ci-dessous montre la structure de la table `employee` telle qu'elle apparaît dans **phpMyAdmin**:

Figure 2: Structure de la table `employee` dans phpMyAdmin

La gestion des données a été facilitée grâce à l'interface intuitive de phpMyAdmin et l'utilisation de **XAMPP** pour configurer le serveur local.

0.4 Résultat Final du Travail Pratique

Avant de détailler les différentes parties du code source, nous présentons ici le résultat final de l'application. Cette étape permettra d'avoir une vue d'ensemble des fonctionnalités réalisées avant d'expliquer en détail leur implémentation.

0.4.1 Fonctionnalités Implémentées

Le projet développé permet la gestion des employés à travers une interface graphique intuitive. Les principales fonctionnalités sont les suivantes :

- **Ajout des employés :** L'utilisateur peut saisir les informations d'un nouvel employé (nom, prénom, poste, date d'embauche, email, salaire) dans le formulaire. En cliquant sur le bouton **Ajouter**, les données sont automatiquement enregistrées dans la base de données `employes`, hébergée sur **localhost** via **phpMyAdmin**.
- **Modification des informations :** En sélectionnant un employé existant dans la liste et en cliquant sur le bouton **Modifier**, les informations de cet employé peuvent être mises à jour. La modification est ensuite sauvegardée dans la base de données.
- **Suppression des employés :** Lorsqu'un utilisateur clique sur le bouton **Supprimer**, une boîte de dialogue s'affiche pour confirmer la suppression. Si la suppression est confirmée, l'enregistrement correspondant est supprimé de la base de données.
- **Affichage des employés :** Le bouton **Afficher** permet de récupérer et de présenter la liste complète des employés enregistrés dans la base de données. Cette liste est affichée sous forme de tableau clair et structuré.

0.4.2 Interface Graphique

L'interface graphique, développée en **Java Swing**, offre une expérience utilisateur fluide et intuitive. Les principaux éléments de l'interface incluent :

- Un formulaire clair pour la saisie des informations (nom, prénom, poste, date d'embauche, email, salaire).
- Des boutons ergonomiques pour chaque action (**Ajouter**, **Modifier**, **Supprimer**, **Afficher**).

- Une table d'affichage des données permettant une visualisation simplifiée des employés enregistrés.

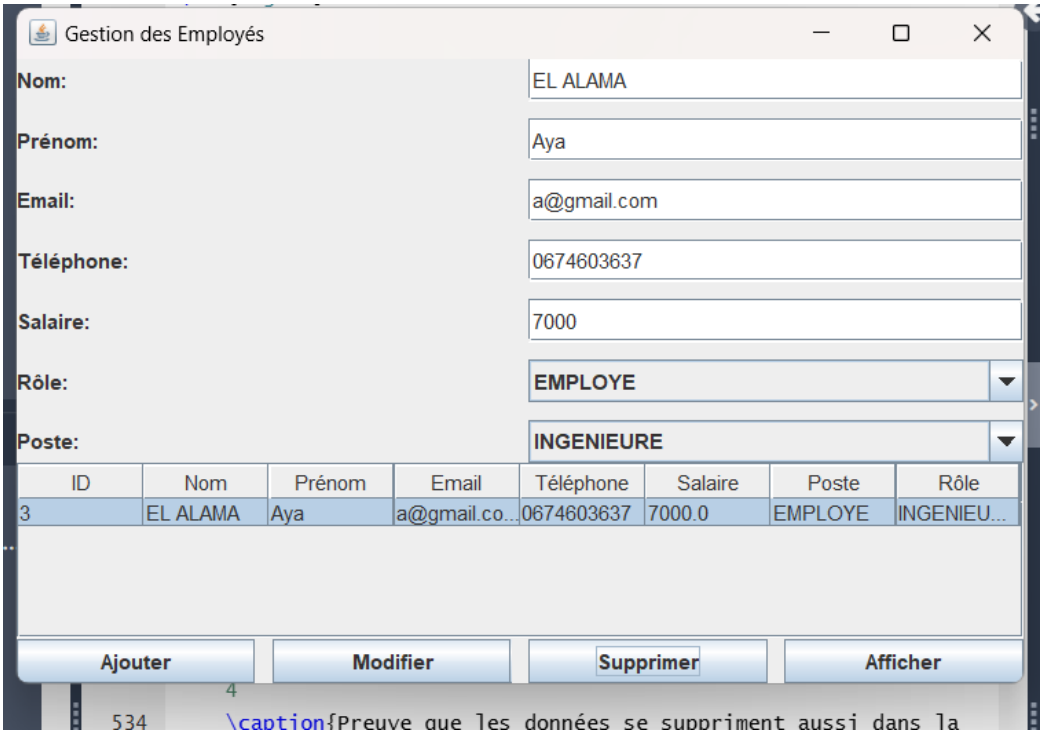

Figure 3: Capture d'écran de l'interface finale de l'application de gestion des employés

0.4.3 Code Source et Explications

Dans les sections suivantes, nous expliquerons en détail :

- La gestion des interactions via les boutons (**Ajouter**, **Modifier**, **Supprimer**, **Afficher**).
- La connexion à la base de données via JDBC.
- Les requêtes SQL utilisées pour manipuler la table `employes`.
- L'architecture MVC adoptée pour structurer le code.

0.5 Présentation du Code Source

Dans cette partie, nous allons détailler les différentes parties du code source. Nous commençons par le `package Model`, qui contient la classe `Employe`. Cette classe est utilisée pour représenter les employés au sein de l'application, en encapsulant leurs attributs et en définissant leurs rôles et postes.

0.5.1 Classe `Employe`

La classe `Employe` est un composant clé du modèle. Elle contient :

- Les attributs de l'employé : `id`, `nom`, `prenom`, `email`, `telephone`, `salaire`, `role`, et `poste`.
- Deux constructeurs pour permettre l'instanciation de la classe avec ou sans `id`.
- Les getters et setters pour accéder et modifier les valeurs des attributs.
- Deux énumérations (`enum`) pour définir les rôles et les postes possibles des employés.

Le code complet de la classe `Employe` est présenté ci-dessous :

```java
package Model;
public class Employe {
    private int id;
    public int getId() {
        return id;
    }




    public void setId(int id) {
        this.id = id;
    }
    public Employe(int id, String nom, String prenom, String email, String telephone, double salaire, Role role,
            Poste poste) {
        super();
        this.id = id;
        this.nom = nom;
        this.prenom = prenom;
        this.email = email;
        this.telephone = telephone;
        this.salaire = salaire;
        this.role = role;
        this.poste = poste;
    }
    private String nom;
    private String prenom;
    private String email;
    private String telephone;
    private double salaire;
    private Role role;
    private Poste poste;

    public Employe(String nom,String prenom,String email,
            String telephone,double salaire,Role role,Poste poste) {
        this.nom=nom;
        this.prenom=prenom;
        this.email=email;
        this.telephone=telephone;
        this.salaire=salaire;
        this.role=role;
        this.poste=poste;
    }
```

Figure 4: Partie 1 - Classe Employe

```
48      public String getNom() {
49          return nom;
50      }
51      public void setNom(String nom) {
52          this.nom=nom;
53      }
54
55
56      public String getPrenom() {
57          return prenom;
58      }
59
60      public String getEmail() {
61          return email;
62      }
63
64      public String getTelephone() {
65          return telephone;
66      }
67
68      public double getSalaire() {
69          return salaire;
70      }
71
72      public Role getRole() {
73          return role;
74      }
75
76      public Poste getPoste() {
77          return poste;
78      }
79
80      public void setPrenom(String prenom) {
81          this.prenom = prenom;
82      }
83
84      public void setEmail(String email) {
85          this.email = email;
86      }
87
88      public void setTelephone(String telephone) {
89          this.telephone = telephone;
90      }
```

Figure 5: Partie 2 - Classe Employe

```
90      }
91
92      public void setSalaire(double salaire) {
93          this.salaire = salaire;
94      }
95
96      public void setRole(Role role) {
97          this.role = role;
98      }
99
00      public void setPoste(Poste poste) {
01          this.poste = poste;
02      }
03
04     public enum Role {
05          ADMIN,
06          EMPLOYE
07          }
08      public enum Poste {
09          INGENIEURE,
10          TEAM_LEADER,
11          PILOTE
12          }
13
14  }
```

Figure 6: Partie 3 - Classe Employe

0.5.2 Classe `EmployeModel` dans le package `Model`

Après avoir présenté la classe `Employe`, nous passons maintenant à la classe `EmployeModel`. Cette classe agit comme un pont entre les données (gérées par `EmployeImpl`) et les opérations logiques nécessaires à la gestion des employés.

Structure et rôle de la classe `EmployeModel`

La classe `EmployeModel` est responsable de la logique métier pour la gestion des employés. Elle effectue des vérifications essentielles avant d'interagir avec la base de données. Elle contient les éléments suivants :

- Un attribut `dao` qui est une instance de `EmployeImpl`, le DAO (Data Access Object) utilisé pour gérer les opérations de la base de données.
- Un constructeur qui initialise le `dao`.
- Une méthode `addEmploye` qui vérifie les données de l'employé avant de l'ajouter à la base de données.

Méthode `addEmploye`

La méthode `addEmploye` joue un rôle crucial dans l'ajout d'un nouvel employé à la base de données. Voici une description des étapes qu'elle effectue :

- Vérifie que le `salaire` est supérieur à 0. Si ce n'est pas le cas, elle affiche un message d'erreur et retourne `false`.
- Valide que l'adresse `email` n'est pas nulle et qu'elle contient le caractère `'@'`. En cas d'erreur, elle affiche un message et retourne `false`.
- Si toutes les validations passent, elle crée un nouvel objet `Employe` à l'aide des paramètres donnés.
- Appelle la méthode `add` du `dao` pour enregistrer le nouvel employé dans la base de données.
- Retourne `true` si l'ajout est réussi.

Capture d'écran du code

Voici une capture d'écran montrant la structure du code de la classe `EmployeModel` :

```java
package Model;

import DAO.EmployeImpl;
import Model.Employe.Poste;

public class EmployeModel {
  private EmployeImpl dao;

  public EmployeModel(EmployeImpl dao) {
    this.dao=dao;
}


public boolean addEmploye(String nom,String pronom,String email,String telephone,double salaire,Model.Employe.Role role,Poste poste) {
    if(salaire<=0) {
        System.out.println("Le salaire doit etre superieur de 0 !!!!!");
       return false;}

    if (email == null || !email.contains("@")) {
        System.out.println("L'email n'est pas valide !");
        return false;
    }

    Employe NvEmploye = new Employe(nom,pronom,email,telephone,salaire,role,poste);
    dao.add(NvEmploye);
return true;

}}
```

Figure 7: Code de la classe `EmployeModel`

Utilité dans l'application

La classe `EmployeModel` est une composante essentielle dans le modèle logique. Elle garantit que les données des employés sont valides avant leur ajout, réduisant ainsi les erreurs dans la base de données. En collaborant avec le DAO `EmployeImpl`, elle assure une gestion fluide des opérations liées aux employés.

Package DAO : Classe Connexion à la base de données

Le package **DAO** (Data Access Object) contient les classes responsables de l'interaction avec la base de données. La classe `connexion` est dédiée à l'établissement et à la fermeture de la connexion avec la base de données MySQL. Elle utilise le JDBC (Java Database Connectivity) pour se connecter à la base de données, permettant à l'application d'effectuer des opérations sur celle-ci.

Nous allons maintenant examiner le code de la classe `connexion`, dont le rôle est de fournir une méthode centralisée pour obtenir et fermer la connexion à la base de données.

Explication du code :

1. **Constantes de connexion :**
- **'url'** : L'URL de connexion à la base de données MySQL, incluant l'adresse de l'hôte ('localhost'), le port ('3306'), et le nom de la base de données ('salma'). - **'user'** : Le nom d'utilisateur utilisé pour la connexion à la base de données (ici, 'root'). - **'password'** : Le mot de passe pour l'utilisateur (ici, vide, car l'utilisateur 'root' n'a pas de mot de passe dans notre configuration locale).

2. **Méthode 'getConnexion()**
- Vérifie si la connexion est déjà établie. Si ce n'est pas le cas, elle crée une nouvelle connexion à l'aide des constantes définies. En cas de succès, un message de confirmation est affiché. En cas d'échec, un message d'erreur est généré.

3. **Méthode 'closeConnexion()**
- Ferme la connexion si elle est ouverte. Si une erreur survient lors de la fermeture, un message d'erreur est affiché.

La classe `connexion` est donc essentielle pour gérer l'accès à la base de données dans l'application. Elle garantit que la connexion à la base de données est centralisée et correctement fermée après utilisation.

```java
package DAO;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class connexion {
    public static final String url = "jdbc:mysql://localhost:3306/salma";
    public static final String user = "root";
    public static final String password = "";
    private static Connection conn = null;

    public static Connection getConnexion() {
        if (conn == null) {
            try {
                conn = DriverManager.getConnection(url, user, password);
                System.out.println("Connexion établie avec succès !");
            } catch (SQLException e) {
                System.out.println("Erreur de connexion !!!!!");

            }
        }
        return conn;
    }

    public static void closeConnexion() {
        if (conn != null) {
            try {
                conn.close();
                conn = null;
                System.out.println("Connexion fermée avec succès !");
            } catch (SQLException e) {
                System.out.println("Erreur lors de la fermeture de la connexion !!!!!");

            }
        }
    }
}
```

Figure 8: Classe Connexion à la base de données

Package DAO : Interface EmployeI

Dans le package **DAO**, l'interface `EmployeI` définit les méthodes nécessaires pour gérer les employés au niveau de la couche de données. Cette interface est une abstraction qui spécifie les opérations principales que toute classe implémentant cette interface doit réaliser. Elle garantit une séparation claire entre la définition et l'implémentation des fonctionnalités.

```java
package DAO;

import java.util.List;

import Model.Employe;
import Model.Employe.Poste;
import Model.Employe.Role;

public interface EmployeI {

Employe findById(int employeId);
List<Employe> findAll();
void add(Employe E);
void update(Employe E,int id);
void delete(int id);
List<Role>findAllRoles();
List<Poste>findAllPostes();


}
```

Figure 9: Interface EmployeI

Explication du code :

1. Méthode findById(int employeId) : Recherche un employé dans la base de données à partir de son identifiant unique et retourne un objet de type `Employe`.

2. Méthode findAll() : Retourne une liste contenant tous les employés enregistrés dans la base de données.

3. Méthode add(Employe E) : Permet d'ajouter un nouvel employé à la base de données en utilisant un objet `Employe` contenant les informations nécessaires.

4. Méthode update(Employe E, int id) : Met à jour les informations d'un employé existant dans la base de données, en utilisant l'identifiant de l'employé (`id`) et un objet `Employe` contenant les nouvelles informations.

5. Méthode delete(int id) : Supprime un employé de la base de données en utilisant son identifiant.

6. Méthode findAllRoles() : Retourne une liste contenant tous les

rôles (`Role`) possibles pour un employé. Cela permet de gérer les rôles de manière dynamique.

7. Méthode findAllPostes() : Retourne une liste contenant tous les postes (`Poste`) possibles pour un employé, facilitant ainsi leur gestion.

Rôle de l'interface :

L'interface `EmployeI` garantit que les opérations principales (CRUD : Create, Read, Update, Delete) et les fonctionnalités additionnelles liées aux rôles et postes des employés sont uniformément définies dans toute classe qui l'implémente.

Package DAO : EmployeImpl

La classe `EmployeImpl`, située dans le package **DAO**, est une implémentation de l'interface `EmployeI`. Elle gère les opérations principales de la base de données pour les employés.

Code source de la classe EmployeImpl :

```java
package DAO;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import Model.Employe;
import Model.Employe.Poste;


public class EmployeImpl implements EmployeI {
    private Connection conn;

    public EmployeImpl() {
        this.conn = connexion.getConnexion();
    }

    @Override
    public void add(Employe E) {
        String Query = "INSERT INTO Employee(nom, prenom, email, telephone, salaire, role, poste) VALUES(?, ?, ?, ?, ?, ?, ?)";

        try (PreparedStatement stmt = conn.prepareStatement(Query)) {
            stmt.setString(1, E.getNom());
            stmt.setString(2, E.getPrenom());
            stmt.setString(3, E.getEmail());
            stmt.setString(4, E.getTelephone());
            stmt.setDouble(5, E.getSalaire());
            stmt.setString(6, E.getRole().name());
            stmt.setString(7, E.getPoste().name());
            stmt.executeUpdate();
            System.out.println("Employé ajouté avec succès !");
        } catch (SQLException e) {
            System.out.println("Erreur lors de l'ajout de l'employé !");
            //e.printStackTrace();

        }
    }
    @Override
```

Figure 10: Classe EmployeIml

```java
    public Employe findById(int employeId) {
        String query = "SELECT * FROM Employee WHERE id = ?";
        try (PreparedStatement stmt = conn.prepareStatement(query)) {
            stmt.setInt(1, employeId);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return new Employe(
                    rs.getString("nom"),
                    rs.getString("prenom"),
                    rs.getString("email"),
                    rs.getString("telephone"),
                    rs.getDouble("salaire"),
                    Employe.Role.valueOf(rs.getString("role")),
                    Poste.valueOf(rs.getString("poste"))
                );
            }
        } catch (SQLException e) {
            System.out.println("Erreur lors de la recherche de l'employé par ID !!!!");
            //e.printStackTrace();
        }
        return null;
    }


@Override
    public List<Employe> findAll() {
        List<Employe> employes = new ArrayList<>();
        String query = "SELECT * FROM Employee";
        try (Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
            while (rs.next()) {
                employes.add(new Employe(
                    rs.getInt("id"),
                    rs.getString("nom"),
                    rs.getString("prenom"),
                    rs.getString("email"),
                    rs.getString("telephone"),
                    rs.getDouble("salaire"),
                    Employe.Role.valueOf(rs.getString("role")),
                    Poste.valueOf(rs.getString("poste"))
                ));
            }
        } catch (SQLException e) {
```

Figure 11: Classe EmployeIml

```
87              System.out.println("Erreur lors de la récupération de tous les employés !!!!!");
88              //e.printStackTrace();
89          }
90          return employes;
91      }
92
93
94  @Override
95  public void update(Employe E, int id) {
96      String query = "UPDATE employee SET nom = ?, prenom = ?, email = ?, telephone = ?, salaire = ?, role = ?, poste = ? WHERE id = ?";
97      try (PreparedStatement stmt = conn.prepareStatement(query)) {
98          stmt.setString(1, E.getNom());
99          stmt.setString(2, E.getPrenom());
100         stmt.setString(3, E.getEmail());
101         stmt.setString(4, E.getTelephone());
102         stmt.setDouble(5, E.getSalaire());
103         stmt.setString(6, E.getRole().name());
104         stmt.setString(7, E.getPoste().name());
105         stmt.setInt(8, id);
106         stmt.executeUpdate();
107         System.out.println("Employe modifier avec succès !");
108     } catch (SQLException e) {
109         System.out.println("Erreur lors de la modefication de l'employe !!!!!");
110         //e.printStackTrace();
111     }
112 }
113
114 @Override
115 public void delete(int id) {
116     String query = "DELETE FROM Employee WHERE id = ?";
117     try (PreparedStatement stmt = conn.prepareStatement(query)) {
118         stmt.setInt(1, id);
119         stmt.executeUpdate();
120         System.out.println("Employe supprimé avec succès !");
121     } catch (SQLException e) {
122         System.out.println("Erreur lors de la suppression de l'employe !!!!");
123         //e.printStackTrace();
124     }
125 }
126
127 @Override
128 public List<Employe.Role> findAllRoles() {
129     return Arrays.asList(Employe.Role.values());
```

Figure 12: Classe EmployeIml

```
130 }
131
132 @Override
133 public List<Poste> findAllPostes() {
134     return Arrays.asList(Poste.values());
135 }
136
137 }
```

Figure 13: Classe EmployeIml

Explications :

La classe `EmployeImpl` implémente toutes les méthodes déclarées dans l'interface `EmployeI`, permettant la gestion des données des employés dans la base de

données.

- **Connexion à la base de données :** Le constructeur de la classe initialise la connexion en appelant la méthode `getConnexion()` de la classe `connexion`.
- **Ajout d'un employé (add) :** Insère un nouvel employé dans la base de données avec une requête `INSERT INTO`. Les champs de l'employé, comme le nom, le prénom, le poste et le rôle, sont passés comme paramètres.
- **Recherche d'un employé (findById) :** Cette méthode utilise une requête SQL pour rechercher un employé par son identifiant (`id`). Si trouvé, un objet `Employe` est retourné.
- **Récupération de tous les employés (findAll) :** Retourne une liste de tous les employés enregistrés dans la base de données à l'aide d'une boucle sur un `ResultSet`.
- **Modification d'un employé (update) :** Permet de modifier les informations d'un employé existant avec une requête `UPDATE`. L'identifiant de l'employé à modifier est passé en paramètre.
- **Suppression d'un employé (delete) :** Supprime un employé dans la base de données à l'aide d'une requête `DELETE` basée sur son identifiant.
- **Gestion des rôles et postes (findAllRoles, findAllPostes) :** Ces méthodes retournent toutes les valeurs possibles des rôles et postes définis dans les énumérations `Employe.Role` et `Poste`. Cela garantit des données cohérentes et normalisées.

Package Controller

Le package `Controller` contient la classe `EmployeController`, qui gère la logique métier entre la vue (`View`) et le modèle (`Model`). Cette classe assure la gestion des actions déclenchées par les boutons de l'interface graphique pour manipuler les employés.

Code source du Controller :

```java
package controller;

import java.util.List;

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

import DAO.EmployeImpl;
import Model.Employe;
import Model.Employe.Poste;
import Model.Employe.Role;

import Model.EmployeModel;
import View.EmployeView;

public class EmployeController {
private EmployeModel model;
private EmployeView view;

public EmployeController(EmployeModel model,EmployeView view) {
    this.model=model;
    this.view=view;
    this.view.btnAjouter.addActionListener(e->addEmploye());
    this.view.btnModifier.addActionListener(e->updateEmploye());
    this.view.btnAfficher.addActionListener(e -> afficherEmploye());
    this.view.btnSupprimer.addActionListener(e -> supprimerEmploye());



}
private void addEmploye() {
    String nom=view.getNom();
    String prenom=view.getPrenom();
    String email=view.getEmail();
    String telephone=view.getTelephone();
    double salaire =view.getSalaire();
    Poste poste=view.getPoste();
    Role role=view.getRole();



    boolean addEmploye=model.addEmploye(nom, prenom, email, telephone,salaire, role, poste);
    if(addEmploye) System.out.println("Employe ajoute avec Succes");
```

Figure 14: La classe `EmployeController`

```
    else System.out.println("Echec d'ajout d'employe !!!!!");
}


private void updateEmploye() {
    int selectedRow = view.table.getSelectedRow();
    int id = (int) view.table.getValueAt(selectedRow, 0);

    String nom=view.getNom();
    String prenom=view.getPrenom();
    String email=view.getEmail();
    String telephone=view.getTelephone();
    double salaire =view.getSalaire();
    Poste poste=view.getPoste();
    Role role=view.getRole();

      Employe employe = new Employe(nom, prenom, email, telephone, salaire, role, poste);
        EmployeImpl employeImpl = new EmployeImpl();

        employeImpl.update(employe,id);







    }
public void afficherEmploye() {
 EmployeImpl employeImpl = new EmployeImpl();
        List<Employe> employes = employeImpl.findAll();

       DefaultTableModel model = (DefaultTableModel) view.table.getModel();
        model.setRowCount(0);

        for (Employe employe : employes) {
            model.addRow(new Object[]{
                employe.getId(),
                employe.getNom(),
                employe.getPrenom(),
                employe.getEmail(),
                employe.getTelephone(),
                employe.getSalaire(),
                employe.getRole(),
                employe.getPoste()
```

Figure 15: La classe `EmployeController`

```
            });
        }
    }
public void supprimerEmploye() {
int selectedRow = view.table.getSelectedRow();
    if (selectedRow == -1) {
        JOptionPane.showMessageDialog(null, "Veuillez sélectionner un employé à supprimer !");
        return;
    }
    int id =view.getId(view.table);

    EmployeImpl employeImpl = new EmployeImpl();

    int confirmation = JOptionPane.showConfirmDialog(null, "Voulez-vous vraiment supprimer cet employé ?", "Confirmation", JOptionPane.YES_NO_OPTION);
    if (confirmation == JOptionPane.YES_OPTION) {
        employeImpl.delete(id);
        JOptionPane.showMessageDialog(null, "Employé supprimé avec succès !");

    }
}
}
```

Figure 16: La classe `EmployeController`

Explications détaillées :

La classe `EmployeController` joue un rôle central dans l'application en connectant la vue et le modèle.

- **Constructeur de la classe :** - Le constructeur initialise les objets `EmployeModel` et `EmployeView`. - Il ajoute des gestionnaires d'événements (`ActionListener`) aux boutons de l'interface utilisateur : `btnAjouter`, `btnModifier`, `btnAfficher`, et `btnSupprimer`.
- **Méthode `addEmploye()`:** - Récupère les données saisies par l'utilisateur via les méthodes `view.getNom()`, `view.getPrenom()`, etc. - Envoie les données au modèle via `model.addEmploye()` pour insertion dans la base de données.
- **Méthode `updateEmploye()`:** - Récupère l'employé sélectionné dans la table (`view.table`). - Met à jour les données de cet employé dans la base via la méthode `employeImpl.update()`.
- **Méthode `afficherEmploye()`:** - Récupère tous les employés via la méthode `employeImpl.findAll()`. - Met à jour le tableau (`DefaultTableModel`) de la vue avec les données récupérées.
- **Méthode `supprimerEmploye()`:** - Récupère l'identifiant de l'employé sélectionné dans le tableau. - Affiche une boîte de dialogue de confirmation avant de supprimer l'employé via la méthode `employeImpl.delete()`.

Package View - Explication

Dans cette section, nous allons examiner le package `View`, qui contient la classe `EmployeView`. Cette classe est responsable de l'affichage de l'interface graphique pour la gestion des employés.

Le code

```java
package View;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import Model.Employe.Role;
import Model.Employe.Poste;

import java.awt.*;
import java.util.ArrayList;

public class EmployeView extends JFrame {

    private JPanel mainPanel, topPanel, centerPanel, bottomPanel;
    private JLabel lblNom, lblPrenom, lblEmail, lblTelephone, lblSalaire, lblPoste, lblRole;
    private JTextField txtNom, txtPrenom, txtEmail, txtTelephone, txtSalaire;
    private JComboBox<Poste> cbPoste;
    private JComboBox<Role> cbRole;
    public JTable table;
    public JButton btnAjouter;
    public JButton btnModifier;
    public JButton btnSupprimer;
    public JButton btnAfficher;

    public EmployeView() {

        setTitle("Gestion des Employés");
        setSize(600, 400);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        mainPanel = new JPanel(new BorderLayout());
        topPanel = new JPanel(new GridLayout(7, 2, 10, 10));
        centerPanel = new JPanel(new BorderLayout());
        bottomPanel = new JPanel(new GridLayout(1, 4, 10, 10));

        lblNom = new JLabel("Nom:");
        lblPrenom = new JLabel("Prénom:");
        lblEmail = new JLabel("Email:");
        lblTelephone = new JLabel("Téléphone:");
        lblSalaire = new JLabel("Salaire:");
        lblPoste = new JLabel("Poste:");
        lblRole = new JLabel("Rôle:");
```

Figure 17: La classe **EmployeView**.

```
44
45          txtNom = new JTextField();
46          txtPrenom = new JTextField();
47          txtEmail = new JTextField();
48          txtTelephone = new JTextField();
49          txtSalaire = new JTextField();
50
51          cbRole = new JComboBox<>(Role.values());
52          cbPoste = new JComboBox<>(Poste.values());
53
54
55
56          table = new JTable(new DefaultTableModel(new Object[]{"ID", "Nom", "Prénom", "Email", "Téléphone", "Salaire", "Poste", "Rôle"},0));
57          JScrollPane scrollPane = new JScrollPane(table);
58
59
60          btnAjouter = new JButton("Ajouter");
61          btnModifier = new JButton("Modifier");
62          btnSupprimer = new JButton("Supprimer");
63          btnAfficher = new JButton("Afficher");
64
65          topPanel.add(lblNom);
66          topPanel.add(txtNom);
67          topPanel.add(lblPrenom);
68          topPanel.add(txtPrenom);
69          topPanel.add(lblEmail);
70          topPanel.add(txtEmail);
71          topPanel.add(lblTelephone);
72          topPanel.add(txtTelephone);
73          topPanel.add(lblSalaire);
74          topPanel.add(txtSalaire);
75          topPanel.add(lblRole);
76          topPanel.add(cbRole);
77          topPanel.add(lblPoste);
78          topPanel.add(cbPoste);
79
80          centerPanel.add(scrollPane, BorderLayout.CENTER);
81
82          bottomPanel.add(btnAjouter);
83          bottomPanel.add(btnModifier);
84          bottomPanel.add(btnSupprimer);
85          bottomPanel.add(btnAfficher);
```

Figure 18: La classe **EmployeView**.

```
87          mainPanel.add(topPanel, BorderLayout.NORTH);
88          mainPanel.add(centerPanel, BorderLayout.CENTER);
89          mainPanel.add(bottomPanel, BorderLayout.SOUTH);
90
91          add(mainPanel);
92          setVisible(true);
93          }
94
95
96      public int getId(JTable table) {
97          int selectedRow = table.getSelectedRow();
98
99          if (selectedRow == -1) {
00              JOptionPane.showMessageDialog(null, "Veuillez sélectionner une ligne !");
01              return -1;
02          }
03          return (int) table.getValueAt(selectedRow, 0);
04
05      }
06
07            public String getNom() {
08               return txtNom.getText();
09              }
10            public String getPrenom() {
11               return txtPrenom.getText();
12               }
13            public String getEmail() {
14                    return txtEmail.getText();
15               }
16            public String getTelephone() {
17               return txtTelephone.getText();
18               }
19            public double getSalaire() {
20               return Double.parseDouble(txtSalaire.getText());
21               }
22            public Role getRole() {
23                Role r=(Role) cbRole.getSelectedItem();
24               return r;
25                }
26            public Poste getPoste() {
27                Poste p=(Poste) cbPoste.getSelectedItem();
28               return p ;
```

Figure 19: La classe **EmployeView**.

```
129             }
130
131     public static void main(String[] args) {
132         new EmployeView();
133     }
134 }
```

Figure 20: La classe **EmployeView**.

Explication du code

Le package `View` contient la classe `EmployeView`, qui est une interface graphique basée sur `JFrame`. Elle utilise plusieurs éléments de l'API Swing, tels que `JTextField`, `JComboBox`, `JTable`, et des panneaux pour organiser l'affichage.

Les éléments de l'interface utilisateur sont regroupés dans trois panneaux principaux : - `topPanel` : Contient les champs de texte pour saisir les informations des employés (nom, prénom, email, téléphone, salaire) ainsi que des menus déroulants pour sélectionner le poste et le rôle. - `centerPanel` : Affiche un tableau (`JTable`) qui liste les employés et leurs informations. Ce tableau est dynamique et permet d'afficher plusieurs employés à la fois. - `bottomPanel` : Contient les boutons d'action (`Ajouter`, `Modifier`, `Supprimer`, `Afficher`) permettant de gérer les employés. Ces boutons sont associés à des actions spécifiques dans le code pour ajouter, modifier ou supprimer des employés dans la base de données.

La méthode principale de cette classe consiste à récupérer les informations des employés à partir des champs de texte et des menus déroulants. Par exemple, la méthode `getNom()` retourne le texte du champ `txtNom`, et la méthode `getRole()` retourne le rôle sélectionné dans le menu déroulant `cbRole`. Le tableau des employés est également mis à jour en temps réel pour afficher les données actuelles.

Les captures d'écran montrent l'organisation et l'initialisation de l'interface graphique. Le code montre la déclaration des variables et l'initialisation des composants graphiqueset montre le tableau des employés avec les boutons d'action pour manipuler ces données.

Package Main - Explication

Dans cette section, nous allons examiner le package `main`, qui contient la classe `main`. Cette classe est le point d'entrée du programme et elle initialise les différents composants du modèle, de la vue et du contrôleur.

Le code

```java
package main;

import DAO.EmployeImpl;
import Model.EmployeModel;
import View.EmployeView;
import controller.EmployeController;

public class main {

    public static void main( String[] args) {
        EmployeView View = new EmployeView();
        EmployeImpl DAO =new EmployeImpl();
        EmployeModel Model = new EmployeModel(DAO);
        new EmployeController(Model,View);

        View.setVisible(true);

    }

}
```

Figure 21: Capture de la classe `main`

Explication du code

Le package `main` contient la classe `main`, qui est responsable de l'initialisation de l'application. Cette classe relie les différents composants de l'architecture MVC (Modèle-Vue-Contrôleur).

Voici un aperçu des actions réalisées dans cette classe :

1. **Initialisation de la Vue (`EmployeView`) :** La vue est initialisée en créant une instance de la classe `EmployeView`, qui contient l'interface graphique pour gérer les employés.

2. **Initialisation du DAO (`EmployeImpl`):** Le DAO (Data Access Object) est initialisé par une instance de la classe `EmployeImpl`. Cette classe

est responsable de la gestion des données des employés, en particulier l'accès à la base de données.

3. **Initialisation du Modèle (`EmployeModel`):** Le modèle est initialisé en créant une instance de `EmployeModel`, qui prend en paramètre l'instance du DAO. Cette classe contient la logique métier de l'application et gère les données des employés en utilisant le DAO pour interagir avec la base de données.

4. **Initialisation du Contrôleur (`EmployeController`):** Enfin, une instance du contrôleur `EmployeController` est créée. Il prend en paramètre le modèle et la vue. Le contrôleur est responsable de la gestion des événements utilisateurs (comme les clics sur les boutons de l'interface) et de l'interaction avec le modèle et la vue pour mettre à jour l'interface graphique en fonction des actions de l'utilisateur.

5. **Affichage de la Vue :** La méthode `setVisible(true)` est appelée sur l'objet `View` pour rendre l'interface graphique visible à l'utilisateur.

La capture d'écran montre la classe `main`, qui configure et démarre l'application en reliant les différentes parties du programme (vue, modèle, contrôleur et DAO).

Résultat du Travail Pratique

Dans cette section, nous allons illustrer les résultats obtenus après avoir exécuté le projet.

Capture d'écran du Résultat

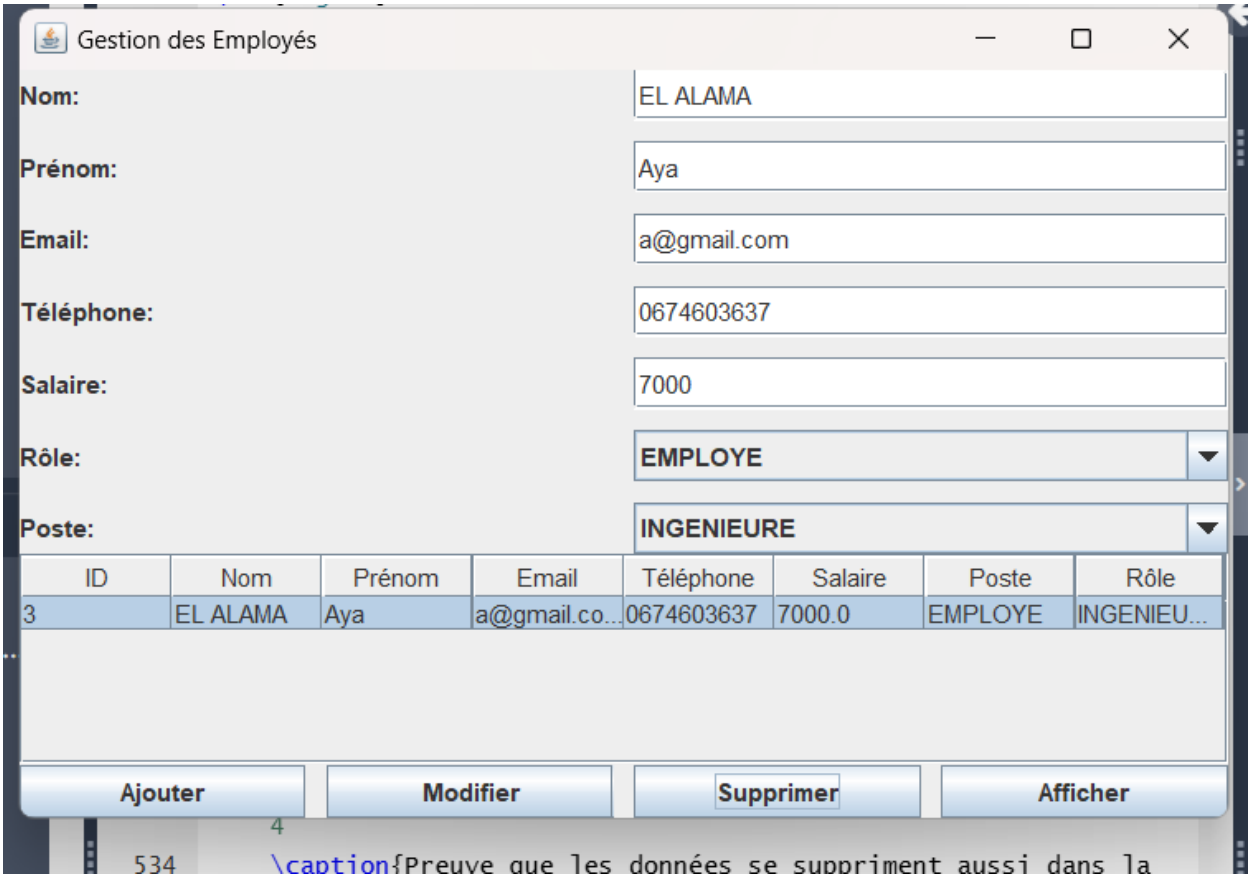

Figure 22: Le Résultat Final

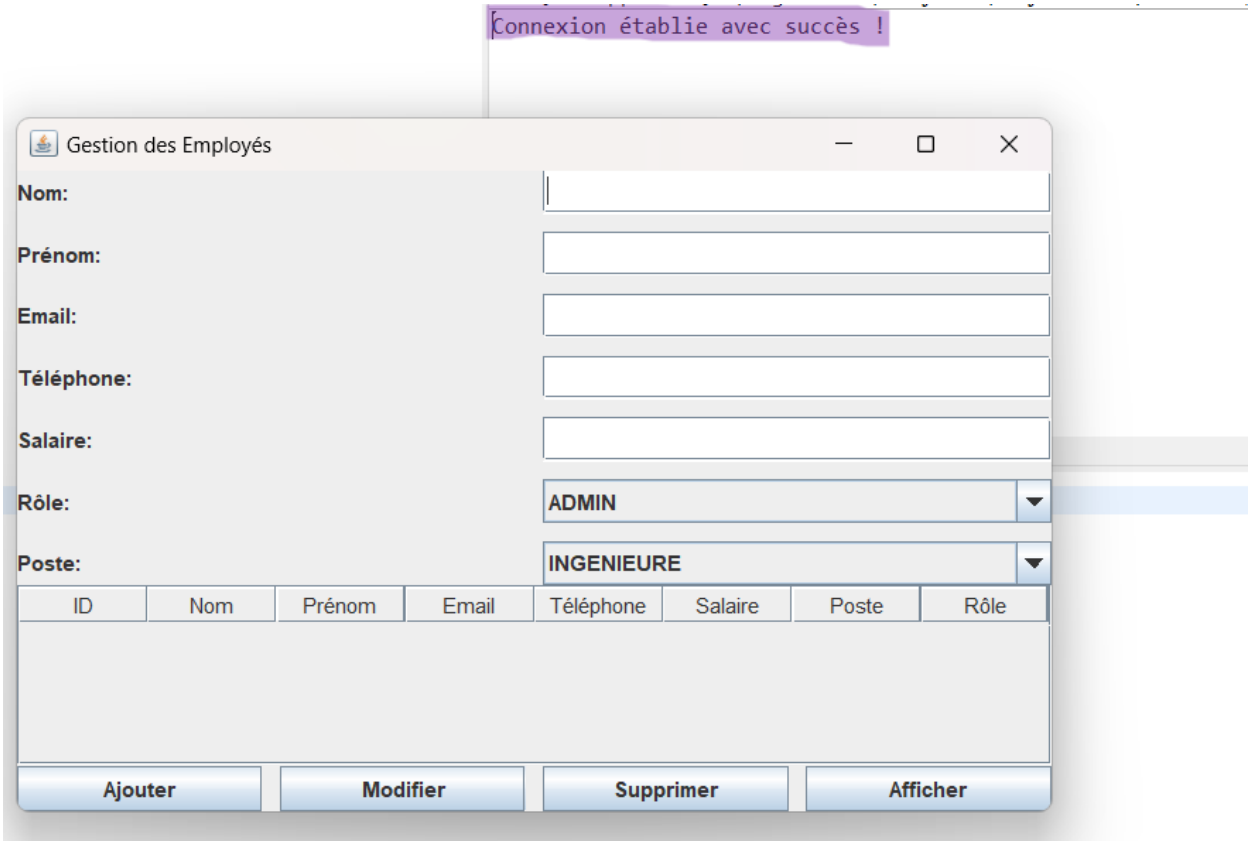

Figure 23: Preuve de la connexion à la Base de Données

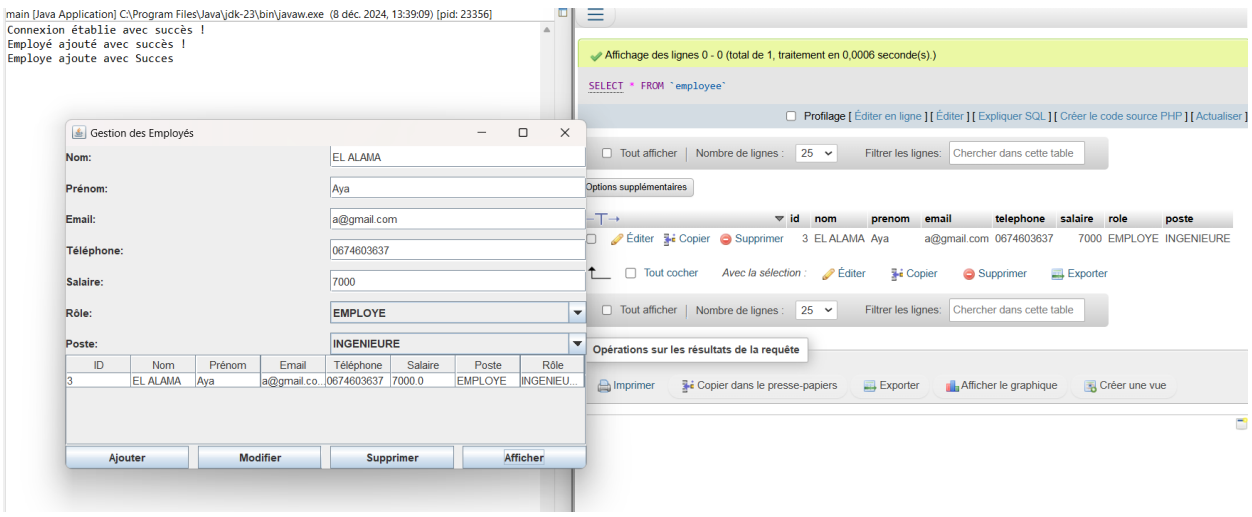

Figure 24: Preuve que les données s'ajoutent dans la BD

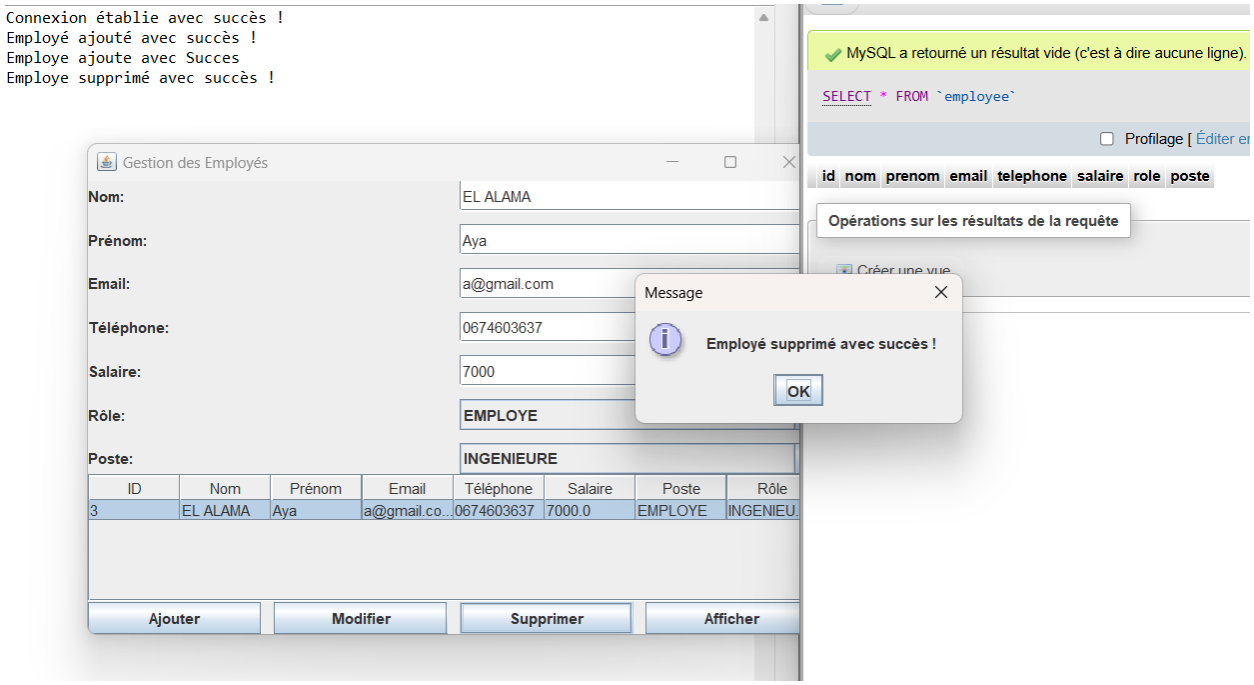

Figure 25: Preuve que les données se suppriment aussi dans la BD

Conclusion

En conclusion, ce projet a permis de mettre en place une application complète de gestion des employés, en utilisant l'architecture MVC (Modèle-Vue-Contrôleur). L'application permet non seulement d'ajouter, de modifier et de supprimer des employés via une interface graphique, mais elle interagit également avec une base de données pour stocker et récupérer les informations des employés.

Ce projet a permis de renforcer mes compétences dans la gestion des données, l'interaction avec les bases de données, et l'implémentation de l'architecture MVC dans une application Java.