# Question 1

* + Après avoir donne une idée du projet ma premier question : Je suis en train de faire la classe Employe avec ses parametres, avec setters et getters sans autres methodes:
    - id\_employe
    - prenom
    - nom
    - poste
    - salaire
    - date\_naissance
    - date\_embauche
    - sexe
    - statut
    - allergies\_preferences\_alimentaires
    - code\_unique

# Réponse 1 :

* class Employe:
* def \_\_init\_\_(self, prenom, nom, poste, salaire, date\_naissance, date\_embauche, sexe, statut, allergies, code\_unique, role):
* self.prenom = prenom
* self.nom = nom
* self.poste = poste
* self.salaire = salaire
* self.date\_naissance = date\_naissance
* self.date\_embauche = date\_embauche
* self.sexe = sexe
* self.statut = statut
* self.allergies = allergies
* self.code\_unique = code\_unique
* self.role = role  # Rôle en tant que numéro
* #def \_\_str\_\_(self):
* #    role\_str = Role.get\_role\_name(self.role)
* #    return f"{self.prenom} {self.nom} - {self.poste} ({self.code\_unique}) - Rôle: {role\_str}"
* # Méthodes pour modifier chaque attribut
* def modifier\_prenom(self, prenom):
* self.prenom = prenom
* def modifier\_nom(self, nom):
* self.nom = nom
* def modifier\_poste(self, poste):
* self.poste = poste
* def modifier\_salaire(self, salaire):
* self.salaire = salaire
* def modifier\_date\_naissance(self, date\_naissance):
* self.date\_naissance = date\_naissance
* def modifier\_date\_embauche(self, date\_embauche):
* self.date\_embauche = date\_embauche
* def modifier\_sexe(self, sexe):
* self.sexe = sexe
* def modifier\_statut(self, statut):
* self.statut = statut
* def modifier\_allergies\_preferences(self, allergies\_preferences):
* self.allergies\_preferences\_alimentaires = allergies\_preferences
* def modifier\_code\_unique(self, code\_unique):
* self.code\_unique = code\_unique
* def modifier\_role(self, role):
* self.role = role  # Méthode pour modifier le rôle

# Nuance

* + J’ai pris la class comme il m’avait envoyé

# Question 2

* + Cette class j'aimerais l'insérer ou modifier directement dans un base de donne erp\_database.db (J’ai envoyé notre class au complet):

# Réponse 2:

* + Classe DAO pour Employe (Data Access Object) :
  + Nous allons maintenant définir une classe EmployeDAO qui gère l'insertion et la mise à jour des employés dans la base de données via DatabaseManager.
* from ERP\_data\_base import DatabaseManager
* class EmployeDAO:
* def \_\_init\_\_(self):
* self.db\_manager = DatabaseManager()
* def get\_employe\_by\_id(self, id\_employe):
* query = "SELECT \* FROM Employes WHERE id\_employe = ?"
* results = self.db\_manager.execute\_query(query, (id\_employe,))
* if results:
* return results[0]  # Retourne le premier enregistrement trouvé
* else:
* return None
* def add\_employe(self, employe\_data):
* query = """
* INSERT INTO Employes (nom, prenom, poste, salaire, date\_naissance, date\_embauche, sexe, statut,
* allergies\_preferences\_alimentaires, code\_unique, role)
* VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
* """
* parameters = (
* employe\_data['nom'],
* employe\_data['prenom'],
* employe\_data['poste'],
* employe\_data['salaire'],
* employe\_data['date\_naissance'],
* employe\_data['date\_embauche'],
* employe\_data['sexe'],
* employe\_data['statut'],
* employe\_data['allergies\_preferences\_alimentaires'],
* employe\_data['code\_unique'],
* employe\_data['role']  # Include the role in the parameters
* )
* rows\_affected = self.db\_manager.execute\_update(query, parameters)
* return rows\_affected > 0
* def update\_employe(self, employe\_data):
* query = """
* UPDATE Employes SET nom = ?, prenom = ?, poste = ?, salaire = ?, date\_naissance = ?,
* date\_embauche = ?, sexe = ?, statut = ?, allergies\_preferences\_alimentaires = ?,
* code\_unique = ?, role = ? WHERE id\_employe = ?
* """
* parameters = (
* employe\_data['nom'],
* employe\_data['prenom'],
* employe\_data['poste'],
* employe\_data['salaire'],
* employe\_data['date\_naissance'],
* employe\_data['date\_embauche'],
* employe\_data['sexe'],
* employe\_data['statut'],
* employe\_data['allergies\_preferences\_alimentaires'],
* employe\_data['code\_unique'],
* employe\_data['role'],  # Include the role in the parameters
* employe\_data['id\_employe']  # Ensure the ID is also passed for the update
* )
* rows\_affected = self.db\_manager.execute\_update(query, parameters)
* return rows\_affected > 0
* def delete\_employe(self, id\_employe):
* query = "DELETE FROM Employes WHERE id\_employe = ?"
* rows\_affected = self.db\_manager.execute\_update(query, (id\_employe,))
* return rows\_affected > 0

# Nuance :

* + Il m’a conseillé de faire un Classe DAO pour Employe, donc j’ai implémenté sa suggestion. Aussi j’ai modifié la classe Employe pour qu’il soit connecté à la base de données.

# Question 3 :

* + Parfait maintenant tu peux m'aider pour afficher dans la vue avec QtWidgets. Quand on click save. Il va créer le nouvel employé et il va l'insérer dans la bd

# Réponse 3:

from PySide6.QtWidgets import (

    QWidget, QVBoxLayout, QLabel, QLineEdit, QPushButton, QComboBox,QMessageBox

)

from ERP\_role import Role

class QAddEmploye(QWidget):

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("Ajouter un Employé")

        layout = QVBoxLayout()

        # Create input fields

        self.prenom\_input = QLineEdit(self)

        self.nom\_input = QLineEdit(self)

        self.poste\_input = QLineEdit(self)

        self.salaire\_input = QLineEdit(self)

        self.date\_naissance\_input = QLineEdit(self)

        self.date\_embauche\_input = QLineEdit(self)

        self.sexe\_input = QLineEdit(self)

        self.statut\_input = QLineEdit(self)

        self.allergies\_input = QLineEdit(self)

        self.code\_unique\_input = QLineEdit(self)

        # Create role selection

        self.role\_input = QComboBox(self)

        self.role\_input.addItem("Administrateur", Role.ROLE\_ADMINISTRATEUR)

        self.role\_input.addItem("HR", Role.ROLE\_HR)

        self.role\_input.addItem("Gérant (Magasin)", Role.ROLE\_GERANT)

        self.role\_input.addItem("Commis", Role.ROLE\_COMMIS)

        # Add input fields to layout

        layout.addWidget(QLabel("Prénom"))

        layout.addWidget(self.prenom\_input)

        layout.addWidget(QLabel("Nom"))

        layout.addWidget(self.nom\_input)

        layout.addWidget(QLabel("Poste"))

        layout.addWidget(self.poste\_input)

        layout.addWidget(QLabel("Salaire par heure"))

        layout.addWidget(self.salaire\_input)

        layout.addWidget(QLabel("Date de naissance"))

        layout.addWidget(self.date\_naissance\_input)

        layout.addWidget(QLabel("Date d'embauche"))

        layout.addWidget(self.date\_embauche\_input)

        layout.addWidget(QLabel("Sexe"))

        layout.addWidget(self.sexe\_input)

        layout.addWidget(QLabel("Statut"))

        layout.addWidget(self.statut\_input)

        layout.addWidget(QLabel("Allergies/Préférences"))

        layout.addWidget(self.allergies\_input)

        layout.addWidget(QLabel("Code Unique"))

        layout.addWidget(self.code\_unique\_input)

        # Add role selection to layout

        layout.addWidget(QLabel("Rôle"))

        layout.addWidget(self.role\_input)

        ajouter\_button = QPushButton("Ajouter Employé", self)

        ajouter\_button.clicked.connect(self.ajouter\_employe)

        layout.addWidget(ajouter\_button)

        self.setLayout(layout)

    def ajouter\_employe(self):

        # Get values from input fields

        employe\_data = {

            'prenom': self.prenom\_input.text(),

            'nom': self.nom\_input.text(),

            'poste': self.poste\_input.text(),

            'salaire': self.salaire\_input.text(),

            'date\_naissance': self.date\_naissance\_input.text(),

            'date\_embauche': self.date\_embauche\_input.text(),

            'sexe': self.sexe\_input.text(),

            'statut': self.statut\_input.text(),

            'allergies\_preferences\_alimentaires': self.allergies\_input.text(),

            'code\_unique': self.code\_unique\_input.text(),

            'role': self.role\_input.currentData()  # Get the selected role

        }

        # Now you can call the method to save this data to the database

        if self.controleur.ajouter\_employe(employe\_data):

            self.clear\_fields()

            QMessageBox.information(self, "Succès", "Employé ajouté avec succès.")

        else:

            QMessageBox.warning(self, "Erreur", "Échec de l'ajout de l'employé.")

    def clear\_fields(self):

        self.prenom\_input.clear()

        self.nom\_input.clear()

        self.poste\_input.clear()

        self.salaire\_input.clear()

        self.date\_naissance\_input.clear()

        self.date\_embauche\_input.clear()

        self.sexe\_input.clear()

        self.statut\_input.clear()

        self.allergies\_input.clear()

        self.code\_unique\_input.clear()

        self.role\_input.setCurrentIndex(0)  # Reset to first role

# Nuance:

* + Après avoir tester sa réponse, vu que tout fonctionne comme je voulais, je l’ai ajouté dans le projet.

# Question 4:

* + Maintenant on doit faire la fenêtre du HR que seulement ont accès les gérants et le gérant global. A l'intérieur de HR il va avoir deux boutons. 1er bouton pour modifier les employés. Ce bouton va ouvrir une autre fenêtre (employe\_hr.py) ou il va avoir un searchbar on peut insérer le codeEmploye et il va chercher dans la bd. Il va l'afficher en cliquant l'employé il va ouvrir un mini fenêtre pour le modifier. Tu peux t'inspirer de cette class pour modifier l'employé: class AddModifyDialog(QDialog

# Réponse 4 :

from PySide6.QtWidgets import QWidget, QPushButton, QHBoxLayout

from PySide6.QtCore import Qt

class qHRWindow(QWidget):

    def \_\_init\_\_(self, parent):

        super().\_\_init\_\_(parent)

        self.setStyleSheet("QPushButton{padding: 10px; background-color:white; border:2px solid black;} QPushButton:pressed{background-color:#cacccf;}")

        # Title and Layout

        self.setWindowTitle("Gestion des Ressources Humaines")

        # Création des boutons

        button1 = QPushButton("Employés")

        button1.clicked.connect(parent.basculer\_vers\_employes\_hr)

        button2 = QPushButton("Commandes")

        button2.clicked.connect(parent.basculer\_vers\_commandes\_hr)

        # Back Button

        back\_button = QPushButton("<-")

        back\_button.clicked.connect(parent.basculer\_before)

         # Mise en page horizontale

        layout = QHBoxLayout()

        layout.addWidget(button1)

        layout.addWidget(button2)

        layout.addWidget(back\_button)

        # Ajouter du padding entre les boutons

        layout.setSpacing(30)  # Espacement entre les boutons

        layout.setContentsMargins(50, 50, 50, 50)  # Marges de la mise en page

        # Centrer le layout dans la fenêtre

        layout.setAlignment(Qt.AlignCenter)

        # Appliquer le layout à la fenêtre

        self.setLayout(layout)

from PySide6.QtWidgets import (

    QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout,

    QLabel, QWidget, QTableWidget, QTableWidgetItem, QAbstractItemView, QMessageBox

)

from employe.ERP\_employeDAO import EmployeDAO

class EmployeHRWindow(QWidget):

    def \_\_init\_\_(self, parent):

        super().\_\_init\_\_(parent)

        # Main layout for Employee interface

        main\_layout = QVBoxLayout()

        # Search Box

        search\_layout = QHBoxLayout()

        self.search\_box = QLineEdit(self)

        self.search\_box.setPlaceholderText("Rechercher employé...")

        self.search\_box.textChanged.connect(self.search\_employee)

        search\_button = QPushButton("Rechercher", self)

        search\_button.clicked.connect(self.search\_employee)

        search\_layout.addWidget(self.search\_box)

        search\_layout.addWidget(search\_button)

        # Labels for additional fields

        self.employee\_code\_label = QLabel("Code Employé:")

        self.hire\_date\_label = QLabel("Date d'Embauche:")

        self.position\_label = QLabel("Position:")

        self.vacation\_days\_label = QLabel("Jours de Vacances Alloués:")

        # Table to display employee information

        self.employee\_table = QTableWidget()

        self.employee\_table.setColumnCount(4)  # Adjust based on required columns

        self.employee\_table.setHorizontalHeaderLabels([

            "Code Employé", "Nom", "Position", "Date d'Embauche"

        ])

        self.employee\_table.setSelectionBehavior(QAbstractItemView.SelectRows)  # Permet de sélectionner une ligne complète

        self.employee\_table.setEditTriggers(QAbstractItemView.NoEditTriggers)  # Empêche l'édition dans la table

        # Connect table row selection

        self.employee\_table.cellClicked.connect(self.on\_employee\_select)

        # Back button to return to the previous window

        back\_button = QPushButton("Back")

        back\_button.clicked.connect(self.vue.basculer\_before)

        # Add widgets to the main layout

        main\_layout.addLayout(search\_layout)

        main\_layout.addWidget(self.employee\_code\_label)

        main\_layout.addWidget(self.hire\_date\_label)

        main\_layout.addWidget(self.position\_label)

        main\_layout.addWidget(self.vacation\_days\_label)

        main\_layout.addWidget(self.employee\_table)

        main\_layout.addWidget(back\_button)  # Adding the back button at the end

        self.setLayout(main\_layout)

    def search\_employee(self):

        search\_text = self.search\_box.text().strip()

        filtered\_employees = self.filter\_employees(search\_text)

        self.update\_employee\_display(filtered\_employees)

    def filter\_employees(self, search\_text):

        employee\_dao = EmployeDAO()  # Assuming you have an EmployeDAO class

        return employee\_dao.get\_employees\_by\_name(search\_text)

    def update\_employee\_display(self, employees):

        self.employee\_table.setRowCount(0)  # Clear previous results

        for employee in employees:

            row\_position = self.employee\_table.rowCount()

            self.employee\_table.insertRow(row\_position)

            self.employee\_table.setItem(row\_position, 0, QTableWidgetItem(employee['code\_unique']))

            self.employee\_table.setItem(row\_position, 1, QTableWidgetItem(f"{employee['prenom']} {employee['nom']}"))

            self.employee\_table.setItem(row\_position, 2, QTableWidgetItem(employee['poste']))

            self.employee\_table.setItem(row\_position, 3, QTableWidgetItem(employee['date\_embauche']))

    def on\_employee\_select(self, row, column):

    # Retrieve the employee code from the first column (Code Employé)

        code\_employe = self.employee\_table.item(row, 0).text()

        # Récupérer les données de l'employé en utilisant son code unique

        employee\_dao = EmployeDAO()

        employee\_data = employee\_dao.get\_employe\_by\_id(code\_employe)

        if employee\_data:

            # Si un employé est trouvé, ouvrir le dialogue de modification

            self.open\_modify\_dialog(employee\_data)

        else:

            # Si aucun employé n'est trouvé, afficher un message d'erreur

            QMessageBox.warning(self, "Erreur", "Aucun employé trouvé avec ce code.")

    def open\_modify\_dialog(self, employee\_data):

        """Open the dialog for modifying the employee."""

        dialog = addModifyDialogEmploye(self, mode="Modifier", product\_data=employee\_data)

        dialog.exec()

from PySide6.QtWidgets import QWidget, QPushButton, QGridLayout, QHBoxLayout, QLabel, QLineEdit

from PySide6.QtCore import Qt

from utils.QListe import QListe  # Importez la classe QListe

class HR\_Commandes(QWidget):

    def \_\_init\_\_(self, parent):

        super().\_\_init\_\_()

        self.parent = parent

        # Create the main layout

        layout = QGridLayout()

        # Création de la barre de recherche

        search\_layout = QHBoxLayout()

        self.search\_bar = QLineEdit()

        self.search\_bar.setPlaceholderText("Rechercher une commande")

        search\_button = QPushButton("Rechercher")

        search\_button.clicked.connect(self.rechercher\_commande)

        back\_button = QPushButton("<-")

        back\_button.clicked.connect(parent.basculer\_before)

        search\_layout.addWidget(self.search\_bar)

        search\_layout.addWidget(back\_button)

        search\_layout.addWidget(search\_button)

        layout.addLayout(search\_layout)

        # Création de la liste des commandes

        self.commande\_liste = QListe("Commandes", [], ["Numéro", "Date", "Client"])

        layout.addWidget(self.commande\_liste)

        self.setLayout(layout)

    def rechercher\_commande(self):

        # Récupération du numéro de commande saisi

        numero\_commande = self.search\_bar.text()

        # Requête pour récupérer les informations de la commande

        # (remplacez par votre requête SQL ou votre méthode de récupération de données)

        commande = self.get\_commande\_from\_db(numero\_commande)

        if commande:

            # Création de la liste des objets pour la classe QListe

            objects = [

                [commande["numero"], commande["date"], commande["client"]]

            ]

            # Mise à jour de la liste des commandes

            self.commande\_liste = QListe("Commandes", objects, ["Numéro", "Date", "Client"])

            self.layout().addWidget(self.commande\_liste)

        else:

            print("Commande non trouvée")

    def get\_commande\_from\_db(self, numero\_commande):

        # Requête pour récupérer les informations de la commande depuis la base de données

        # (remplacez par votre requête SQL ou votre méthode de récupération de données)

        query = "SELECT \* FROM commandes WHERE numero = ?"

        # Exécution de la requête et récupération des résultats

        # (remplacez par votre méthode d'exécution de requête et de récupération de résultats)

        result = {"numero": numero\_commande, "date": "2022-01-01", "client": "Client 1"}

        return result

# Nuance:

* + J’ai implemente hr.py, il y a un problème avec employe\_hr.py donc j’ai pas touche commandes\_hr.py