Soutien AI :

Alek :

# Question 1

Avec Python pyside6 qt, peux tu me faire une page avec trois boutton sur la gauche nommé : ouvrir, modifier, suprimer; sur la droite une liste qui contient 4 colonne nommé : nom,prenom, username, poste; et puis me faire un exmple d'incertion dans la liste

## réponse

import sys

from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QTableWidget, QTableWidgetItem, QHeaderView

from PySide6.QtCore import Qt

class MainWindow(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

# Initialisation de la fenêtre principale

self.setWindowTitle("Gestion des Employés")

self.setGeometry(100, 100, 800, 400)

# Layout principal

main\_layout = QHBoxLayout()

# Partie gauche (les boutons)

self.sidebar = QVBoxLayout()

# Boutons à gauche

self.ouvrir\_button = QPushButton("Ouvrir")

self.modifier\_button = QPushButton("Modifier")

self.supprimer\_button = QPushButton("Supprimer")

# Ajout des boutons dans la sidebar

self.sidebar.addWidget(self.ouvrir\_button)

self.sidebar.addWidget(self.modifier\_button)

self.sidebar.addWidget(self.supprimer\_button)

# Partie droite (la table)

self.table = QTableWidget()

# Initialisation de la table avec 4 colonnes

self.table.setColumnCount(4)

self.table.setHorizontalHeaderLabels(["Nom", "Prénom", "Username", "Poste"])

# Autres configurations de la table

self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

# Ajouter des données initiales

self.insert\_initial\_data()

# Ajout des éléments à la fenêtre principale

main\_layout.addLayout(self.sidebar)

main\_layout.addWidget(self.table)

self.setLayout(main\_layout)

def insert\_initial\_data(self):

# Exemple d'insertion dans la table

data = [

("Dupont", "Jean", "jdupont", "Développeur"),

("Martin", "Sophie", "smartin", "Responsable"),

("Lemoine", "Paul", "plemoine", "Analyste"),

("Durand", "Claire", "cdurand", "Directeur")

]

# Insertion des données dans la table

for row\_idx, row\_data in enumerate(data):

self.table.insertRow(row\_idx)

for col\_idx, value in enumerate(row\_data):

self.table.setItem(row\_idx, col\_idx, QTableWidgetItem(value))

# Lancer l'application

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

## soutien:

bon blueprint de page pour commencer ma liste avec des exemple

# Question 2

J’aimerais une fonction qui me permettrait d’aller chercher le nom de mes colonnes dans ma base de données :

## Réponse

db\_manager = DatabaseManager('erp\_database.db')

column\_query = "PRAGMA table\_info(Succursales);"

columns\_info = db\_manager.execute\_query(column\_query)

## soutien :

utile pour demander toutes les info d’une table (ajouter des champs)

# Question 3

J’aimerais une système d’historique pour les pages pour pouvoir mettre un bouton back qui reviens a la page précédente

## Réponse

Dans le constructeur : self.history = []

Quand tu change de page : self.history.append(self.stacked\_widget.currentWidget())

Quand tu veux revenir en arrière : if self.history:

previous\_widget = self.history.pop() # Retirer le dernier widget visité

self.stacked\_widget.setCurrentWidget(previous\_widget)

## soutien :

Permet de revenir en arrière, pouvoir revenir a différente page selon on vient de ou

# Question 4

J’aimerais changer une table pour lui ajouter un nouveau champ, j’ai un champ input\_nom qui est le nom du champ et un input input\_type qui contient le type

## Réponse

def ajouter(self):

# Vérifiez que les champs sont remplis

if self.input\_nom.text() != "" and self.input\_type.currentText() != "":

nom\_table = self.mode

nom\_new\_champ = self.input\_nom.text()

type\_new\_champ = self.input\_type.currentText()

db\_manager = DatabaseManager('erp\_database.db')

query = f"ALTER TABLE {nom\_table} ADD COLUMN {nom\_new\_champ} {type\_new\_champ};"

try:

# Exécution de la requête

db\_manager.execute\_update(query)

print(f"Colonne '{nom\_new\_champ}' ajoutée à la table '{nom\_table}'.")

except sqlite3.Error as e:

print(f"Erreur lors de l'ajout de la colonne : {e}")

## soutien :

Permet de faire la page ajouter champ

# Question 5

J’aimerais une fonction qui vérifie si ma table Employes est vide

## Réponse

def is\_table\_empty(self):

count = self.db\_manager.execute\_query("SELECT COUNT(\*) FROM Employes")

return count[0][0] == 0 # Retourne True si la table est vide

## soutien :

Permet de vérifier pour la première connexion si un employé existe

# Question 6

Qu’est ce qui serait une bonne manière dans mon erp de toujours savoir je suis dans quel succursale, quel est mon rôle, etc..

## Réponse

Tu devrais te faire une classe que tu peux appeler de partout pour savoir ou tu es ou qui tu es : class Emplacement:

# Attribut de classe (variable globale)

succursalesCode = 0

employeID = 0

## soutien :

Permet d’afficher les employé le stock de la succursale ou je suis

# Question 7

voici un exemple de ma base de données pour aller chercher un employé et afficher son info, j'aimerais une fonction qui va chercher tout mes employé et m'affcihe toutes leur informations : try: db\_manager = DatabaseManager('erp\_database.db') query = "SELECT \* FROM Employes WHERE id\_employe = ?" results = db\_manager.execute\_query(query, (1,)) for row in results: nom = row['nom'] prenom = row['prenom'] print(f"Nom : {nom}, Prénom : {prenom}") except sqlite3.Error as e: print(f"Une erreur est survenue : {e}")

## Réponse

try: db\_manager = DatabaseManager('erp\_database.db') # Chemin de votre base de données # Requête SQL pour récupérer tous les employés query = "SELECT nom, prenom, username, poste FROM Employes" results = db\_manager.execute\_query(query) if results: # Affichage des résultats dans la console avec print print(f"{'Nom':<20} {'Prénom':<20} {'Username':<20} {'Poste':<20}") print("-" \* 80) # Séparateur pour améliorer la lisibilité for row in results: nom, prenom, username, poste = row print(f"{nom:<20} {prenom:<20} {username:<20} {poste:<20}") else: print("Aucun employé trouvé dans la base de données.")

## soutien :

Quand je veux vérifier l’information de mes employé

Yosef :

# Question 1

Pour mon programme ERP, je voudrais créer une classe Singleton pour la base de données qui permet d'aller chercher ou de mettre des données dans la base de données depuis n'importe où dans le code, sans avoir besoin de créer un nouvel objet. Penses-tu que ça fait du sens ?

Réponse :  
Utiliser un **singleton** pour la gestion de la base de données dans votre programme ERP peut effectivement faire sens dans certains cas. Le **modèle singleton** garantit qu'il n'y a qu'**une seule instance** de la classe à travers toute l'application, ce qui permet d'accéder à la base de données depuis n'importe où sans avoir à créer de nouvelles instances.

**Avantages :**

* **Accès globalisé** : Vous pouvez accéder à la base de données depuis n'importe quelle partie de votre code sans passer d'objets en paramètres.
* **Gestion centralisée** : Toutes les opérations de base de données passent par une seule instance, ce qui facilite la maintenance et les mises à jour.
* **Contrôle des ressources** : Limite le nombre de connexions simultanées à la base de données.

**Cependant, il y a des points importants à considérer :**

1. **Sécurité des threads** :
   * Si votre application est multithreadée, vous devez vous assurer que votre singleton est **thread-safe** pour éviter les problèmes de concurrence.
   * Des mécanismes de synchronisation ou l'utilisation de bibliothèques adaptées peuvent être nécessaires.
2. **Scalabilité et Performance** :
   * Une seule instance peut devenir un **goulot d'étranglement** si de nombreuses requêtes sont effectuées simultanément.
   * Cela peut affecter les performances de votre application.
3. **Gestion des Connexions** :
   * Les bases de données modernes utilisent souvent des **pools de connexions** pour gérer efficacement les connexions multiples.
   * Un singleton peut ne pas tirer pleinement parti de ces optimisations.
4. **Tests et Maintenance** :
   * Les singletons peuvent compliquer les **tests unitaires** car ils introduisent des dépendances globales difficiles à isoler.
   * Cela peut rendre votre code moins flexible et plus difficile à maintenir.
5. **Couplage Fort** :
   * L'utilisation d'un singleton peut entraîner un **couplage fort** entre les différentes parties de votre application, réduisant la modularité.

**Alternatives à considérer :**

* **Injection de Dépendances (DI)** :
  + Utiliser un conteneur d'injection de dépendances pour gérer les instances de vos classes.
  + Cela facilite le **test unitaire** et améliore la flexibilité du code.
* **Patron de Conception Factory** :
  + Utiliser une **factory** pour gérer la création des instances de connexion, permettant plus de contrôle.
* **Gestionnaire de Connexions** :
  + Implémenter un gestionnaire qui utilise un **pool de connexions**, optimisant l'accès concurrent à la base de données.

**Conclusion :**

Créer une classe singleton pour la base de données peut être une solution pratique pour des applications simples ou lorsque le contrôle strict des instances est nécessaire. Cependant, pour une application ERP plus complexe, il est crucial de considérer les implications en termes de **performance**, **scalabilité** et **maintenabilité**.

Je vous recommande d'évaluer les besoins spécifiques de votre projet et de considérer éventuellement des architectures plus modulaires qui facilitent les tests et la maintenance à long terme.

Question 2 :  
À l’aide de mes tables pourrait-tu créer les méthodes create table au complet :

## Réponse :

Bien sûr ! Je vais vous fournir la méthode \_create\_tables complète pour créer toutes vos tables dans la classe DatabaseManager en Python avec SQLite. Cette méthode inclura les instructions SQL pour créer chacune des tables de votre projet ERP, adaptées pour fonctionner avec SQLite.

**1. Méthode \_create\_tables Complète**

Voici la méthode \_create\_tables avec toutes les instructions SQL nécessaires :

python

Copy code

def \_create\_tables(self):

# Création de la table Employes

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Employes (

id\_employe INTEGER PRIMARY KEY AUTOINCREMENT,

prenom TEXT NOT NULL,

nom TEXT NOT NULL,

poste TEXT,

salaire REAL,

date\_naissance TEXT,

date\_embauche TEXT,

sexe TEXT CHECK(sexe IN ('M', 'F')),

statut TEXT,

allergies\_preferences\_alimentaires TEXT,

code\_unique TEXT

)

""")

# Création de la table Roles (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Roles (

id\_role INTEGER PRIMARY KEY AUTOINCREMENT,

nom\_role TEXT NOT NULL,

description TEXT

)

""")

# Création de la table Employes\_Roles (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Employes\_Roles (

id\_employe\_role INTEGER PRIMARY KEY AUTOINCREMENT,

id\_employe INTEGER NOT NULL,

id\_role INTEGER NOT NULL,

date\_debut TEXT,

date\_fin TEXT,

FOREIGN KEY (id\_employe) REFERENCES Employes(id\_employe),

FOREIGN KEY (id\_role) REFERENCES Roles(id\_role)

)

""")

# Création de la table Succursales

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Succursales (

id\_succursale INTEGER PRIMARY KEY AUTOINCREMENT,

nom TEXT NOT NULL,

gerant INTEGER,

adresse TEXT,

telephone TEXT,

code TEXT,

date\_ouverture TEXT,

statut TEXT,

FOREIGN KEY (gerant) REFERENCES Employes(id\_employe)

)

""")

# Création de la table Employes\_Succursales

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Employes\_Succursales (

id\_employe\_succursale INTEGER PRIMARY KEY AUTOINCREMENT,

id\_employe INTEGER NOT NULL,

id\_succursale INTEGER NOT NULL,

date\_debut TEXT,

date\_fin TEXT,

FOREIGN KEY (id\_employe) REFERENCES Employes(id\_employe),

FOREIGN KEY (id\_succursale) REFERENCES Succursales(id\_succursale)

)

""")

# Création de la table Horaires

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Horaires (

id\_horaire INTEGER PRIMARY KEY AUTOINCREMENT,

id\_employe INTEGER NOT NULL,

id\_succursale INTEGER NOT NULL,

date TEXT,

heure\_entree TEXT,

heure\_sortie TEXT,

statut TEXT,

FOREIGN KEY (id\_employe) REFERENCES Employes(id\_employe),

FOREIGN KEY (id\_succursale) REFERENCES Succursales(id\_succursale)

)

""")

# Création de la table Clients

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Clients (

id\_client INTEGER PRIMARY KEY AUTOINCREMENT,

nom TEXT NOT NULL,

prenom TEXT NOT NULL,

adresse TEXT,

telephone TEXT,

email TEXT,

date\_inscription TEXT,

statut TEXT,

notes TEXT

)

""")

# Création de la table Fournisseurs

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Fournisseurs (

id\_fournisseur INTEGER PRIMARY KEY AUTOINCREMENT,

nom TEXT NOT NULL,

contact\_nom TEXT,

contact\_telephone TEXT,

email TEXT,

adresse TEXT,

date\_inscription TEXT,

statut TEXT,

notes TEXT

)

""")

# Création de la table Produits

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Produits (

id\_produit INTEGER PRIMARY KEY AUTOINCREMENT,

code\_produit TEXT UNIQUE,

nom\_produit TEXT NOT NULL,

prix REAL,

description TEXT

)

""")

# Création de la table Stocks

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Stocks (

id\_stock INTEGER PRIMARY KEY AUTOINCREMENT,

id\_produit INTEGER NOT NULL,

id\_succursale INTEGER NOT NULL,

qte\_actuelle INTEGER,

qte\_max INTEGER,

qte\_min\_restock INTEGER,

FOREIGN KEY (id\_produit) REFERENCES Produits(id\_produit),

FOREIGN KEY (id\_succursale) REFERENCES Succursales(id\_succursale)

)

""")

# Création de la table Fournisseurs\_Produits

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Fournisseurs\_Produits (

id\_fournisseur\_produit INTEGER PRIMARY KEY AUTOINCREMENT,

id\_fournisseur INTEGER NOT NULL,

id\_produit INTEGER NOT NULL,

prix\_achat REAL,

delai\_livraison INTEGER,

FOREIGN KEY (id\_fournisseur) REFERENCES Fournisseurs(id\_fournisseur),

FOREIGN KEY (id\_produit) REFERENCES Produits(id\_produit)

)

""")

# Création de la table Categories (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Categories (

id\_categorie INTEGER PRIMARY KEY AUTOINCREMENT,

nom\_categorie TEXT NOT NULL,

description TEXT

)

""")

# Création de la table Produits\_Categories (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Produits\_Categories (

id\_produit\_categorie INTEGER PRIMARY KEY AUTOINCREMENT,

id\_produit INTEGER NOT NULL,

id\_categorie INTEGER NOT NULL,

FOREIGN KEY (id\_produit) REFERENCES Produits(id\_produit),

FOREIGN KEY (id\_categorie) REFERENCES Categories(id\_categorie)

)

""")

# Création de la table Commandes

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Commandes (

id\_commande INTEGER PRIMARY KEY AUTOINCREMENT,

id\_client INTEGER,

date\_commande TEXT,

statut TEXT,

total REAL,

mode\_paiement TEXT,

date\_livraison TEXT,

adresse\_livraison TEXT,

notes TEXT,

FOREIGN KEY (id\_client) REFERENCES Clients(id\_client)

)

""")

# Création de la table Commandes\_Produits

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Commandes\_Produits (

id\_commande\_produit INTEGER PRIMARY KEY AUTOINCREMENT,

id\_commande INTEGER NOT NULL,

id\_produit INTEGER NOT NULL,

quantite INTEGER,

prix\_unitaire REAL,

total\_ligne REAL,

FOREIGN KEY (id\_commande) REFERENCES Commandes(id\_commande),

FOREIGN KEY (id\_produit) REFERENCES Produits(id\_produit)

)

""")

# Création de la table Achats

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Achats (

id\_achat INTEGER PRIMARY KEY AUTOINCREMENT,

id\_fournisseur INTEGER,

date\_achat TEXT,

statut TEXT,

total REAL,

notes TEXT,

FOREIGN KEY (id\_fournisseur) REFERENCES Fournisseurs(id\_fournisseur)

)

""")

# Création de la table Achats\_Produits

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Achats\_Produits (

id\_achat\_produit INTEGER PRIMARY KEY AUTOINCREMENT,

id\_achat INTEGER NOT NULL,

id\_produit INTEGER NOT NULL,

quantite INTEGER,

prix\_unitaire REAL,

total\_ligne REAL,

FOREIGN KEY (id\_achat) REFERENCES Achats(id\_achat),

FOREIGN KEY (id\_produit) REFERENCES Produits(id\_produit)

)

""")

# Création de la table Paiements (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Paiements (

id\_paiement INTEGER PRIMARY KEY AUTOINCREMENT,

id\_commande INTEGER,

montant REAL,

date\_paiement TEXT,

mode\_paiement TEXT,

statut TEXT,

notes TEXT,

FOREIGN KEY (id\_commande) REFERENCES Commandes(id\_commande)

)

""")

# Création de la table Promotions (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Promotions (

id\_promotion INTEGER PRIMARY KEY AUTOINCREMENT,

code\_promo TEXT UNIQUE,

description TEXT,

pourcentage\_remise REAL,

date\_debut TEXT,

date\_fin TEXT,

statut TEXT

)

""")

# Création de la table Commandes\_Promotions (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Commandes\_Promotions (

id\_commande\_promotion INTEGER PRIMARY KEY AUTOINCREMENT,

id\_commande INTEGER NOT NULL,

id\_promotion INTEGER NOT NULL,

FOREIGN KEY (id\_commande) REFERENCES Commandes(id\_commande),

FOREIGN KEY (id\_promotion) REFERENCES Promotions(id\_promotion)

)

""")

# Création de la table Logistiques (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Logistiques (

id\_logistique INTEGER PRIMARY KEY AUTOINCREMENT,

id\_commande INTEGER NOT NULL,

statut\_livraison TEXT,

transporteur TEXT,

numero\_suivi TEXT,

date\_expedition TEXT,

date\_livraison TEXT,

notes TEXT,

FOREIGN KEY (id\_commande) REFERENCES Commandes(id\_commande)

)

""")

# Création de la table Retours (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Retours (

id\_retour INTEGER PRIMARY KEY AUTOINCREMENT,

id\_commande INTEGER NOT NULL,

date\_retour TEXT,

motif TEXT,

statut TEXT,

notes TEXT,

FOREIGN KEY (id\_commande) REFERENCES Commandes(id\_commande)

)

""")

# Création de la table Comptabilite (Optionnelle)

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS Comptabilite (

id\_transaction INTEGER PRIMARY KEY AUTOINCREMENT,

date\_transaction TEXT,

type\_transaction TEXT,

montant REAL,

description TEXT,

id\_commande INTEGER,

id\_achat INTEGER,

FOREIGN KEY (id\_commande) REFERENCES Commandes(id\_commande),

FOREIGN KEY (id\_achat) REFERENCES Achats(id\_achat)

)

""")

# Commit des changements

self.connection.commit()

Question 3 :  
J’ai le problème suivant quand j’Essaie de push :  
$ git push fatal: The current branch yosef has no upstream branch. To push the current branch and set the remote as upstream, use git push --set-upstream origin yosef

Réponse:  
**Bonjour,**

Le message d'erreur que vous rencontrez lors de l'exécution de git push est :

vbnet

Copy code

fatal: The current branch yosef has no upstream branch.

To push the current branch and set the remote as upstream, use

git push --set-upstream origin yosef

**Explication du Problème :**

* **Contexte :** Vous travaillez sur une branche locale nommée yosef qui n'existe pas encore sur le dépôt distant (souvent appelé origin).
* **Cause de l'Erreur :** Git ne sait pas vers quelle branche distante il doit pousser vos modifications, car la branche yosef n'est pas configurée pour suivre une branche distante correspondante.

**Solution Proposée :**

Pour résoudre ce problème, vous devez pousser votre branche locale yosef vers le dépôt distant origin et configurer Git pour que la branche locale suive automatiquement la branche distante correspondante.

**Étapes à Suivre :**

1. **Utiliser la Commande Suggérée par Git :**

Exécutez la commande suivante :

bash

Copy code

git push --set-upstream origin yosef

**Ou en abrégé :**

bash

Copy code

git push -u origin yosef

**Ce que fait cette commande :**

* + **git push :** Pousse vos modifications vers le dépôt distant.
  + **--set-upstream ou -u :** Configure la branche locale pour suivre la branche distante correspondante.
  + **origin :** Le nom du dépôt distant (par défaut, origin).
  + **yosef :** Le nom de la branche que vous souhaitez pousser.

1. **Vérifier que le Poussage a Réussi :**

Après avoir exécuté la commande, vous devriez voir un message confirmant que la branche a été créée sur le dépôt distant et que le suivi est configuré.

**Exemple de message de succès :**

csharp

Copy code

Branch 'yosef' set up to track remote branch 'yosef' from 'origin'.

1. **Poursuivre le Travail sur la Branche :**

Désormais, vous pouvez continuer à travailler sur la branche yosef, et les futures commandes git push et git pull se référeront automatiquement à origin/yosef.

**Informations Supplémentaires :**

* **Pourquoi Configurer une Branche de Suivi ?**

Configurer une branche locale pour suivre une branche distante facilite la synchronisation entre votre travail local et le dépôt distant. Git saura automatiquement vers quelle branche pousser ou tirer les modifications.

* **Vérifier le Statut des Branches :**

Vous pouvez utiliser la commande suivante pour vérifier quelles branches locales suivent quelles branches distantes :

bash

Copy code

git branch -vv

* **Changer le Suivi d'une Branche Existante :**

Si vous souhaitez modifier la branche distante suivie par une branche locale existante, vous pouvez utiliser :

bash

Copy code

git branch --set-upstream-to=origin/branche\_distante

**Conseils pour le Travail en Équipe :**

* **Rester Synchronisé :** Assurez-vous de tirer (git pull) régulièrement les modifications du dépôt distant pour éviter les conflits.
* **Communication :** Informez votre équipe que vous avez créé une nouvelle branche yosef sur le dépôt distant, surtout si plusieurs personnes travaillent sur le projet.

**Résumé :**

Le message d'erreur vous indique simplement que Git a besoin de savoir où pousser votre branche locale yosef. En exécutant la commande git push --set-upstream origin yosef, vous créez la branche sur le dépôt distant et configurez votre branche locale pour suivre la branche distante correspondante.

**N'hésitez pas à me faire savoir si vous avez d'autres questions ou si vous rencontrez des difficultés supplémentaires !**

Question 4 :  
Maintenant je voudrais que tu fasses la page des fournisseurs selon la logique suivante :  
fournisseur bouton fournisseur cliquer sur le bouton fournisseur passage sur la page fournisseur affiche la table fournisseur affichage des differents fournisseur Ajouter fournisseur clique sur ajouter affichage de l'onglet ajouter fourniseur demande de remplir chaque colonne de la table fournisseur(si jamais on rajoute des champs) prend les champs de la tables fournisseur afficher un bouton ajouter et annuler clique sur ajouter lance une fonction pour ajouter le nouveau fournisseur a la table sql ajoute un fournisseurs a la table clique sur annuler reviens a la page d'avant modifier fournisseur cliquer sur le bouton modifier passage en mode selection clique dans la liste de fournisseur celui qu'on veut changer affichage de l'onglet modifier fournisseur affiche le même ecran que ajouter mais les information du fournisseur sont affichée dans les champs en allant les chercher dans la base de données prend les champs de la tables fournisseur afficher un bouton modifier et annuler clique sur modifier lance une fonction pour modifier le fournisseur dans la table sql modifie le valeur choisi de la table Fournisseur clique sur annuler reviens a la page d'avant crée une commande clique sur le bouton créer commande affichage de l'onglet commande affiche les champs nom produit, quantité et prix affichage du bouton cree une commande et annuler clique sur cree une commande ajoute un commande a la tables Commandes crée une commande clique sur annuler reviens en arriere afficher commande fournisseur clique sur le bouton afficher commande affichage de l'onglet Afficher commande affiche automatiquement les commandes faites au fournisseur affiche les commandes de la table Commandes clique sur une commande affiche les détails de la commande sélectionner  
  
Réponse :  
**1. Création de la Classe QFournisseur**

Commençons par créer la classe QFournisseur, qui sera le widget principal pour la gestion des fournisseurs.

python

Copy code

# fournisseur.py

from PySide6.QtWidgets import (

QWidget, QPushButton, QLabel, QLineEdit, QTableWidget,

QTableWidgetItem, QVBoxLayout, QHBoxLayout, QGridLayout, QMessageBox

)

from PySide6.QtCore import Qt

from add\_modify\_supplier\_dialog import AddModifySupplierDialog

from create\_order\_dialog import CreateOrderDialog

class QFournisseur(QWidget):

def \_\_init\_\_(self, parent, db\_manager):

super().\_\_init\_\_()

self.parent = parent

self.db\_manager = db\_manager

# Create the main layout

layout = QGridLayout()

# Left-side buttons layout

add\_button = QPushButton("Ajouter")

modify\_button = QPushButton("Modifier")

create\_order\_button = QPushButton("Créer Commande")

show\_orders\_button = QPushButton("Afficher Commandes")

back\_button = QPushButton("<-")

button\_layout = QVBoxLayout()

button\_layout.addWidget(back\_button)

button\_layout.addWidget(add\_button)

button\_layout.addWidget(modify\_button)

button\_layout.addWidget(create\_order\_button)

button\_layout.addWidget(show\_orders\_button)

layout.addLayout(button\_layout, 0, 0)

# Title

title\_label = QLabel("Gestion des Fournisseurs")

title\_label.setAlignment(Qt.AlignCenter)

layout.addWidget(title\_label, 0, 1)

# Search bar

search\_label = QLabel("Rechercher :")

self.search\_input = QLineEdit()

self.search\_input.textChanged.connect(self.search\_suppliers)

search\_layout = QHBoxLayout()

search\_layout.addWidget(search\_label)

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

layout.addLayout(search\_layout, 1, 1)

# Suppliers table

self.supplier\_table = QTableWidget()

self.supplier\_table.setColumnCount(5) # Ajustez le nombre de colonnes selon vos besoins

self.supplier\_table.setHorizontalHeaderLabels(

["ID", "Nom", "Adresse", "Téléphone", "Email"]

)

layout.addWidget(self.supplier\_table, 2, 1)

# Set layout

self.setLayout(layout)

# Load data

self.load\_data()

# Connect button actions to methods

add\_button.clicked.connect(self.add\_supplier)

modify\_button.clicked.connect(self.modify\_supplier)

create\_order\_button.clicked.connect(self.create\_order)

show\_orders\_button.clicked.connect(self.show\_orders)

back\_button.clicked.connect(parent.basculer\_vers\_splash)

def load\_data(self):

try:

query = "SELECT \* FROM Fournisseurs"

results = self.db\_manager.execute\_query(query)

self.supplier\_table.setRowCount(0) # Clear existing data

for row\_number, row\_data in enumerate(results):

self.supplier\_table.insertRow(row\_number)

self.supplier\_table.setItem(row\_number, 0, QTableWidgetItem(str(row\_data['id\_fournisseur'])))

self.supplier\_table.setItem(row\_number, 1, QTableWidgetItem(row\_data['nom']))

self.supplier\_table.setItem(row\_number, 2, QTableWidgetItem(row\_data.get('adresse', '')))

self.supplier\_table.setItem(row\_number, 3, QTableWidgetItem(row\_data.get('telephone', '')))

self.supplier\_table.setItem(row\_number, 4, QTableWidgetItem(row\_data.get('email', '')))

except Exception as e:

QMessageBox.warning(self, "Erreur", f"Une erreur s'est produite lors du chargement des données : {e}")

def add\_supplier(self):

dialog = AddModifySupplierDialog(self.db\_manager, mode="Ajouter")

dialog.supplier\_added.connect(self.insert\_supplier\_into\_db)

dialog.exec\_()  
  
Question 5 :  
Pour créer une commande :  
J'ai mis la classe Commande dans son propre fichier. Le souci, c'est que maintenant, quand je veux créer une commande, j'obtiens cette erreur :  
AttributeError: 'QFournisseur' object has no attribute 'CreateOrderDialog'.  
  
Réponse :  
Bonjour,

Je comprends que vous rencontrez l'erreur suivante lorsque vous essayez de créer une commande :

csharp

Copy code

AttributeError: 'QFournisseur' object has no attribute 'CreateOrderDialog'

Cette erreur se produit parce que votre classe CreateOrderDialog n'est plus une classe interne de QFournisseur. Vous l'avez déplacée dans son propre fichier, ce qui est une bonne pratique pour améliorer la lisibilité et la maintenance du code. Cependant, cela signifie que vous devez ajuster la façon dont vous faites référence à cette classe dans votre code.

**Cause de l'Erreur**

Dans votre classe QFournisseur, vous utilisez self.CreateOrderDialog pour créer une instance de la boîte de dialogue de création de commande :

python

Copy code

dialog = self.CreateOrderDialog(self.db\_manager, supplier\_id)

Lorsque CreateOrderDialog était une classe interne à QFournisseur, cette référence était correcte. Cependant, maintenant que vous avez déplacé CreateOrderDialog dans son propre fichier, elle n'est plus accessible via self.CreateOrderDialog.

Par conséquent, l'attribut CreateOrderDialog n'existe pas dans l'objet QFournisseur, ce qui entraîne l'erreur AttributeError: 'QFournisseur' object has no attribute 'CreateOrderDialog'.

**Solution**

Pour résoudre ce problème, vous devez :

1. **Importer la classe CreateOrderDialog dans le fichier où est définie QFournisseur.**
2. **Modifier les références à self.CreateOrderDialog pour utiliser simplement CreateOrderDialog.**