Sujet :

Dans le cadre de notre projet de SAÉ « Conception et implémentation d'une base de données », nous avons travaillé sur la modélisation et la gestion informatique des données de la coopérative de sel marin de l'île de Ré. Cette structure regroupe les sauniers locaux et centralise leurs activités d'approvisionnement et de vente. Notre objectif était de représenter leur système d'information à travers une base de données relationnelle, d'automatiser l'import des données réelles fournies en fichiers tableur, et de construire une interface graphique simple avec Python/Tkinter.

Organisation :

Nous avons d'abord commencé par une lecture attentive du sujet et des annexes pour comprendre les entités clés et leurs relations. Cela nous a permis de dégager un ensemble d'entités : **SAUNIER**, **PRODUIT**, **ENTREE**, **CLIENT**, **SORTIE**, **CONCERNER**, et une nouvelle entité **PRIX**, introduite pour prendre en compte les tarifs par année et par type de produit.

Avec ces éléments, nous avons construit un **modèle conceptuel de données (MCD)** à l'aide du logiciel Looping. Nous avons veillé à bien définir les cardinalités, les attributs obligatoires, les contraintes d'unicité, et à respecter la convention de ne pas mettre d'accents dans les noms. Le fichier produit, selmarin.loo, représente cette première étape. Le MCD a été complété manuellement pour intégrer la gestion des prix, en s'appuyant sur les exemples fournis dans l'annexe B du sujet.

Une fois le MCD établi, nous avons généré le **modèle logique de données (MLD)**, puis le **script SQL de création de la base**, nommé selmarin_create.sql. Ce script contient les instructions pour créer toutes les tables et les relations entre elles. Nous avons ensuite créé la base selmarin-tda dans MySQL via PhpMyAdmin (sous EasyPHP), et importé ce script pour vérifier la création effective des tables et des liaisons, en utilisant la vue concepteur.

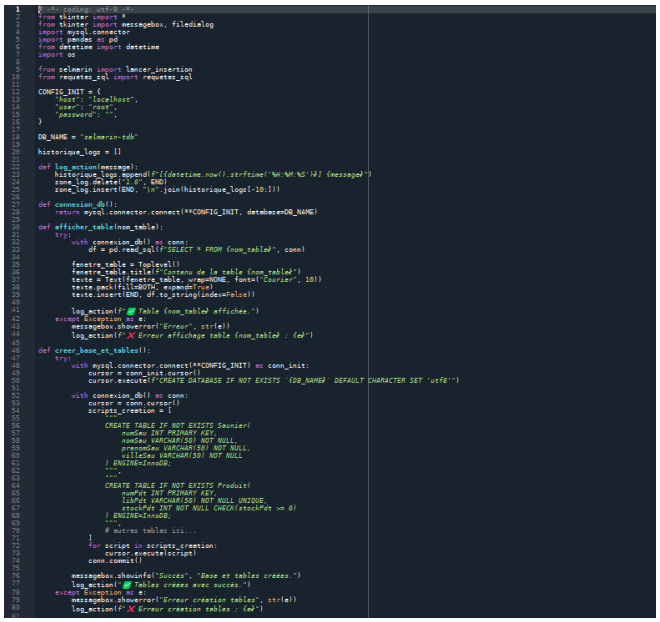
Nous avons ensuite rempli cette base avec des données de test issues des annexes. Ces insertions ont été regroupées dans un script SQL selmarin_insert.sql, où chaque table a été alimentée de manière structurée. Ces requêtes ont été testées une à une dans l'interface PhpMyAdmin pour s'assurer qu'elles s'exécutaient sans erreurs et respectaient les contraintes des clés étrangères.

Pour automatiser l'intégration des **données complètes de l'année 2023**, nous avons développé un **script Python nommé selmarin.py**. Ce script utilise le module csv pour ouvrir les fichiers fournis (au format .csv), traite ligne par ligne les données, et les insère dynamiquement dans la base MySQL. Ce travail a nécessité un traitement minutieux des types de données, la gestion d'erreurs potentielles (valeurs manquantes ou mal formatées), et la construction d'un système de correspondance entre les entêtes CSV et les colonnes SQL.

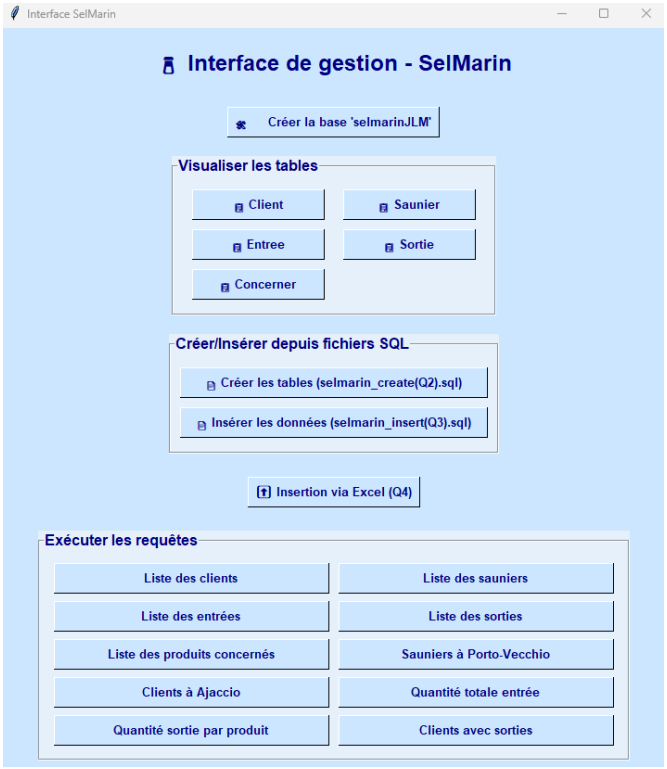
```python
import mysql.connector
from datetime import datetime


def transformer_format_date(date_str):
    try:
        return datetime.strptime(date_str, "%d/%m/%Y")
    except ValueError:
        return date_str


def est_une_date(valeur):
    try:
        datetime.strptime(valeur, "%d/%m/%Y")
        return True
    except ValueError:
        return False


def decouper_et_convertir(ligne_texte, index_colonnes):
    elements = ligne_texte.strip().split(";")
    selection = [elements[i] for i in index_colonnes]
    resultat = []
    for champ in selection:
        if est_une_date(champ):
            resultat.append(transformer_format_date(champ))
        elif champ.isdigit():
            resultat.append(int(champ))
        else:
            resultat.append(champ)
    return tuple(resultat)


def charger_fichier_csv(nom_fichier, indices_utiles):
    with open(nom_fichier, mode="r", encoding="ISO-8859-1") as fichier:
        lignes_csv = fichier.readlines()[1:]  # Saut de l'en-tête
        return [decouper_et_convertir(l, indices_utiles) for l in lignes_csv]


def executer_insertion(liste_donnees, sql_requete, connexion_bdd, curseur_bdd):
    for enregistrement in liste_donnees:
        try:
            curseur_bdd.execute(sql_requete, enregistrement)
            connexion_bdd.commit()
            print(f"Insertion réussie : {enregistrement}")
        except Exception as erreur:
            print(f"Erreur lors de l'insertion {enregistrement} : {erreur}")
            connexion_bdd.rollback()


def etablir_connexion():
    infos_connexion = {
        "host": "localhost",
        "user": "root",
        "password": "",
        "database": "selmarinJLM"
    }
    return mysql.connector.connect(**infos_connexion)


def lancer_traitement():
    connexion = etablir_connexion()
    curseur = connexion.cursor()

    instructions = {
        "client": ("client.csv", "INSERT INTO Client VALUES (%s, %s, %s, %s)", [0, 1, 2, 3]),
        "entree": ("entree.csv", "INSERT INTO Entree(numEnt,dateEnt,qteEnt,numPdt,numSau) VALUES (%s, %s, %s, %s, %s)", [0, 1, 2, 3, 4]),
        "saunier": ("saunier.csv", "INSERT INTO Saunier VALUES (%s, %s, %s, %s)", [0, 1, 2, 3]),
        "sortie": ("sortie.csv", "INSERT INTO Sortie(numSort,dateSort,numCli) VALUES (%s, %s, %s)", [0, 1, 2]),
        "concerner": ("sortie.csv", "INSERT INTO Concerner(numSort,numPdt,qteSort) VALUES (%s, %s, %s)", [0, 3, 4]),
    }

    for nom, (fichier_csv, requete_sql, colonnes) in instructions.items():
        print(f"Lecture en cours : {fichier_csv}")
        donnees_lues = charger_fichier_csv(fichier_csv, colonnes)

        executer_insertion(donnees_lues, requete_sql, connexion, curseur)

    curseur.close()
    connexion.close()
```

Après avoir importé l'ensemble des données, nous avons développé **10 requêtes SQL** destinées à aider un responsable de la coopérative à prendre des décisions. Ces requêtes incluent des sélections avec conditions, des groupements, des agrégations, une différence algébrique (exclusion), des requêtes d'insertion, de mise à jour ou de suppression, ainsi que la création et la réutilisation d'une **vue**. Toutes ces requêtes sont regroupées dans le fichier selmarin_requetes.sql, et ont été testées individuellement.

Nous avons ensuite construit une **interface graphique en Python avec Tkinter**, servant de tableau de bord central pour interagir avec la base de données. Cette interface permet d'accéder aux fonctionnalités suivantes : visualisation des tables principales, lancement des 10 requêtes préparées, et insertion directe des données issues du fichier Excel.



Pour rendre l'application plus complète, nous avons intégré des retours visuels (pop-ups d'erreur ou de confirmation), et organisé les widgets de manière à ce que l'interface reste claire et intuitive. Toutefois, l'automatisation complète de la création de la base (depuis Tkinter) n'a pas été implémentée entièrement, car cela aurait nécessité un travail supplémentaire de gestion des connexions et des droits utilisateur dans MySQL.



Ce projet a représenté un défi à plusieurs niveaux : compréhension de la structure logique des données, rigueur dans la modélisation, manipulation précise des formats CSV, et construction d'une interface fonctionnelle. L'aspect technique a permis de mettre en pratique les connaissances acquises en SQL, Python et algorithmique. Du point de vue personnel, ce travail a été très formateur car il permet de voir la cohérence d'un projet informatique complet, depuis la modélisation jusqu'à l'exploitation finale par une interface.

Si certaines fonctionnalités ont parfaitement fonctionné (requêtes, visualisation, import CSV), d'autres auraient mérité un approfondissement, notamment l'automatisation complète ou l'ajout de visualisations graphiques (histogrammes, courbes de stock). Une amélioration future envisageable serait l'ajout d'un espace de connexion sécurisé, ou l'intégration d'un export des résultats sous format PDF ou Excel.

En somme, cette SAÉ a été l'occasion d'acquérir une **vision concrète du cycle de vie d'un projet de base de données**, tout en renforçant nos compétences en programmation Python, en modélisation et en gestion de données relationnelles.