Rapport Labo 1 - LOG430 - Louis Thevenet

Avant-propos

J'ai retiré les tests `test_user_select` et `test_product_select` car ils présupposaient l'existence d'éléments dans la base de données, de plus, les tests d'insertion utilisent déjà `select_all` pour vérifier que l'insertion a réussi.

J'ai aussi ajouté une fonction de nettoyage de la base de données après les tests.

Question 1

Quelles commandes avez-vous utilisées pour effectuer les opérations UPDATE et DELETE dans MySQL ? Avez-vous uniquement utilisé Python ou également du SQL ? Veuillez inclure le code pour illustrer votre réponse.

J'ai utilisé les commandes SQL suivantes:

- `self.cursor.execute("UPDATE users SET email=%s,name=%s WHERE id=%s", (user.email, user.name, user.id))`
- `self.cursor.execute("DELETE FROM users WHERE id=%s", (user_id, ))`

Le code python utilise des curseurs pour exécuter ces commandes. Voici le code complet des fonctions `update` et `delete`:

```python
def update(self, user):
    """ Update given user in MySQL """
      self.cursor.execute("UPDATE users SET email=%s,name=%s WHERE id=%s",
                          (user.email, user.name, user.id))
      self.conn.commit()

def delete(self, user_id):
    """ Delete user from MySQL with given user ID """
    self.cursor.execute("DELETE FROM users WHERE id=%s", (user_id, ))
    self.conn.commit()
    if self.cursor.fetchone():
        return True
    return False
```

J'ai également ajouté ces actions dans l'application:

```
log430-a25-labo1 on  main [$!?] via 🐍 v3.13.6 via ❄️  impure (nix-shell-env)
❯ sudo docker compose exec store_manager_app python ./src/store_manager.py
===== LE MAGASIN DU COIN =====
/app/.env

1. Montrer la liste d'utilisateurs
2. Ajouter un utilisateur
3. Mettre à jour un utilisateur
4. Supprimer un utilisateur
5. Quitter l'application
Choisissez une option:
```

Question 2

Quelles commandes avez-vous utilisées pour effectuer les opérations dans MongoDB ? Avez-vous uniquement utilisé Python ou également du SQL ? Veuillez inclure le code pour illustrer votre réponse.

MongoDB permet de n'utiliser que des commandes Python pour manipuler les données. J'ai utilisé les méthodes Python suivantes pour les opérations `UPDATE` et `DELETE`:

- `self.users.update_one({"_id": id}, {"$set": {"email": user.email, "name": user.name, }})`
- `self.users.delete_one({"_id": id})`

En voici les implémentations complètes:

```
def update(self, user):
    """Update user in Mongo. """
    if type(user.id) is str:
        id = ObjectId(user.id)
    else:
        id = user.id
    result = self.users.update_one(
        {"_id": id}, {"$set": {
            "email": user.email,
            "name": user.name,
        }})
    return result.matched_count > 0

def delete(self, user_id):
    """ Delete user from Mongo with given user ID """
    if type(user_id) is str:
        id = ObjectId(user_id)
    else:
        id = user_id
    result = self.users.delete_one({"_id": id})
    print(result.deleted_count)
    return result.deleted_count > 0
```

Pour garder la compatilibité avec les entrées utilisateur données

Question 3

Comment avez-vous implémenté votre product_view.py ? Est-ce qu'il importe directement la ProductDAO ? Veuillez inclure le code pour illustrer votre réponse.

J'ai implémenté le fichier `product_view.py` en suivant le modèle MVC (Modèle-Vue-Contrôleur). Le fichier `product_view.py` n'importe pas directement `ProductDAO`. Il importe `ProductController`, qui gère la logique métier et les interactions avec le modèle de données via `ProductDAO`. code

```
from models.product import Product
from controllers.product_controller import ProductController


class ProductView:

    @staticmethod
    def show_options():
        """ Show menu with operation options which can be selected by the user
"""
        controller = ProductController()
        while True:
            print(
                "\n1. Montrer la liste des produits\n2. Ajouter un produit\n3.
Quitter l'application"
            )
            choice = input("Choisissez une option: ")

            if choice == '1':
                products = controller.list_products()
                ProductView.show_products(products)
```

```
            elif choice == '2':
                name, brand, price = ProductView.get_inputs()
                product = Product(None, name, brand, price)
                controller.create_product(product)
            elif choice == '3':
                controller.shutdown()
                break
            else:
                print("Cette option n'existe pas.")

    @staticmethod
    def show_products(products):
        """ List products """
        print("\n".join(
            f"{prod.id}: {prod.name} ({prod.brand}) - {prod.price}€"
            for prod in products))

    @staticmethod
    def get_inputs():
        """ Prompt user for inputs necessary to add a new product """
        name = input("Nom du produit : ").strip()
        brand = input("Marque du produit : ").strip()
        price = float(input("Prix du produit : ").strip())
        return name, brand, price
```

J'ai également fait une vue principale `main_view.py` qui importe `ProductView` et `UserView` et agit comme un sur-menu.

```
class MainView:

    @staticmethod
    def show_options():
        """ Show main menu with operation options for users and products """
        while True:
            print("\n===== MENU PRINCIPAL =====\n"
                  "1. Gestion des utilisateurs\n"
                  "2. Gestion des produits\n"
                  "3. Quitter l'application")
            choice = input("Choisissez une option: ")

            if choice == '1':
                print("\n===== GESTION DES UTILISATEURS =====")
                UserView.show_options()
            elif choice == '2':
                print("\n===== GESTION DES PRODUITS =====")
                ProductView.show_options()
            elif choice == '3':
                print("Au revoir!")
                break
            else:
                print("Cette option n'existe pas.")
```

Question 4

Si nous devions créer une application permettant d'associer des achats d'articles aux utilisateurs (Users → Products), comment structurerions-nous les données dans MySQL par rapport à MongoDB ?

Dans MySQL, nous utiliserions des tables relationnelles avec des clés étrangères pour représenter les associations entre les utilisateurs et les produits. Par exemple, nous aurions une table `users`, une table `products`, et une table `purchases` qui contiendrait des références aux identifiants des utilisateurs et des produits, ainsi qu'une date d'achat.

Cela permettrait de maintenir l'intégrité des données et de faciliter les requêtes complexes grâce aux jointures. Mais l'évolution serait plus compliquée est nécessiterait des migrations.

Dans MongoDb, nous pourrions utiliser des documents imbriqués ou des références . Par exemple, chaque utilisateur pourrait contenir un tableau de références aux produits achetés.

Cette approche est plus flexible et permet de stocker des données hiérarchiques, mais elle peut compliquer les requêtes si les relations deviennent trop complexes.