TP gestion et audit de transactions dans PostgreSQL

A rendre à la fin de la semaine. A faire en groupe (2/3 personnes max). Vous pouvez prendre votre projet comme exemple d'utilisation. Si ce n'est pas fini, vous montrerez comment vous intégrerez tout ca dans le projet.

Objectifs du Projet

Objectif général

Développer une application qui utilise une base de données PostgreSQL pour gérer les transactions d'un système de gestion de stock, avec un accent sur l'intégrité, la concurrence et l'audit des données.

Objectifs spécifiques

1. Conception et modélisation de base de données: Créer un schéma de base de données pour un système de gestion de stock qui prend en charge les transactions concurrentes et l'audit.
2. Développement d'application: Développer une application en Python qui utilise psycopg2 ou SQLAlchemy pour interagir avec la base de données PostgreSQL.
3. Gestion des transactions: Mettre en œuvre des transactions pour garantir l'intégrité des données lors des opérations de stock.
4. Audit et sécurité: Configurer et utiliser `pgaudit` pour auditer les transactions et assurer la sécurité des données.

Partie Développement

Exigences techniques

1. Création de la base de données et tables:
   - Utiliser PostgreSQL pour créer une base de données `stock_management`.
   - Créer des tables nécessaires telles que `products`, `transactions`, etc.
2. Application de gestion de stock:
   - Écrire une application en Python pour gérer le stock.
   - Implémenter des fonctions pour ajouter, retirer, et mettre à jour le stock.
   - S'assurer que l'application gère correctement les transactions concurrentes.
3. Implémentation de l'audit:
   - Configurer `pgaudit` pour auditer toutes les modifications apportées à la table `transactions`.
   - Stocker les logs d'audit pour analyse future.

Exigences de conception

- Concevoir une interface utilisateur intuitive pour interagir avec le système.
- Prévoir des mécanismes de gestion d'erreur et de récupération en cas de défaillance.

Partie rédactionnelle/réflexion

Exigences rédactionnelles

1. Documentation du système:
   - Rédiger une documentation technique qui explique le fonctionnement de l'application et les choix de conception.
   - Inclure des diagrammes UML ou ER pour illustrer la conception de la base de données.
2. Analyse de l'audit:
   - Fournir un rapport sur l'importance de l'audit dans les systèmes de gestion de stock et comment `pgaudit` aide à atteindre cela.
3. Sécurité des données:
   - Écrire une section sur les meilleures pratiques de sécurité à adopter dans le contexte de la base de données PostgreSQL.

- Discuter de l'impact de la concurrence sur la sécurité et l'intégrité des données.

Exigences de réflexion

- Réfléchir sur les défis rencontrés lors du développement et comment ils ont été surmontés.
- Analyser l'efficacité des transactions et des mécanismes d'audit mis en place.
- Proposer des améliorations ou des fonctionnalités supplémentaires pour étendre le projet.

Livrables

1. Application Python complète avec code source et instructions de déploiement.
2. Documentation technique du projet.
3. Rapport d'audit et de sécurité détaillé.
4. Rapport de réflexion sur les apprentissages et les défis du projet.

Évaluation

Le projet sera évalué sur les critères suivants :

- Fonctionnalité et robustesse de l'application.
- Description de la mise en place des outils et des tests de fonctionnalités
- Clarté et détail de la documentation technique.
- Compréhension et application des concepts d'audit et de sécurité.
- Profondeur de la réflexion et de l'analyse dans le rapport final.

Annexe

Exemple de code Python pour simuler l'application

Générer par ChatGPT. Je n'ai pas vérifié le code! Le fichier ```database.py

```
import psycopg2
from psycopg2 import pool

class Database:
    _connection_pool = None

    @staticmethod
    def initialise(**kwargs):
        Database._connection_pool = psycopg2.pool.SimpleConnectionPool(1, 10, **kwargs)

    @staticmethod
    def get_connection():
        return Database._connection_pool.getconn()

    @staticmethod
    def return_connection(connection):
        Database._connection_pool.putconn(connection)

    @staticmethod
    def close_all_connections():
        Database._connection_pool.closeall()

    @staticmethod
    def execute_query(query, params=None, commit=False):
        connection = Database.get_connection()
        with connection.cursor() as cursor:
            cursor.execute(query, params)
            if commit:
                connection.commit()
            return cursor.fetchall()
```

```
        finally:
            Database.return_connection(connection)
```

Fichier `transactions.py` :

```
from database import Database

def add_stock(product_id, amount):
    Database.execute_query('UPDATE products SET stock = stock + %s WHERE id = %s', (amount,
product_id), commit=True)

def remove_stock(product_id, amount):
    Database.execute_query('UPDATE products SET stock = stock - %s WHERE id = %s', (amount,
product_id), commit=True)

def create_transaction(product_id, amount, transaction_type):
    Database.execute_query('INSERT INTO transactions (product_id, amount, transaction_type)
VALUES (%s, %s, %s)',
                           (product_id, amount, transaction_type), commit=True)
```

Le main pour exécuter des transactions. Fichier `app.py` .

```
from database import Database
import transactions

def main():
    # Initialise database connection
    Database.initialise(user='username', password='password', database='stock_management',
host='localhost')

    # Example operations
    transactions.add_stock(1, 50)
    transactions.remove_stock(1, 20)
    transactions.create_transaction(1, 20, 'OUTGOING')

    # Close database connection
    Database.close_all_connections()

if __name__ == '__main__':
    main()
```