TRANSACTION ET VERROUILLAGES

Les transactions et vérrouillages Sont des concepts en gestion de base de données, pour assurer la cohérence des données dans un environnement où plusieurs utilisateurs ou processus peuvent accéder nsimultanément à la meme base de données.

TRANSACTIONS

Une transaction un ensemble d'instructions SQL qui doivent être exécutées ensemble ou pas du tout.

Propriétés ACID

Une transaction possède 4 proprités appelées `ACID` :

1. `Atomicité` : une transaction est une unité indivisible, soit toutes les instructions sont exécutées, soit aucune.
2. `Cohérence` : une transaction doit laisser la base de données dans un état cohérent.
3. `Isolation` : une transaction doit être isolée des autres transactions.
4. `Durabilité` : une transaction doit être durable, c'est-à-dire que les modifications apportées par une transaction doivent être persistantes.

Création d'une base de données `compte_bancaire` et de deux tables `transactions` et `compte` :

```sql
CREATE DATABASE compte_bancaire; -- Création de la base de données
```

```sql
USE compte_bancaire; -- Utilisation de la base de données
```

Création des tables `transactions` et `compte` :

```sql
CREATE TABLE transactions (
    id INT AUTO_INCREMENT,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE compte (
    id INT AUTO_INCREMENT,
    id_transaction INT,
    montant DECIMAL(10, 2),
    PRIMARY KEY (id),
    FOREIGN KEY (id_transaction) REFERENCES transactions(id)
```

```
);
```

Insertion des données dans les tables `transactions` et `compte` :

```
INSERT INTO transactions (nom, prenom, email)
VALUES ('DelPierro', 'Alessandro', 'alessandro@gmail.com'),
       ('Meloni', 'Giorgia', 'meloni@gmail.com');

INSERT INTO compte (id_transaction, montant) VALUES (1, 1000), (2, 1500);
```

Affichage des données des tables `transactions` et `compte` :

```
SELECT * FROM transactions;

SELECT * FROM compte;
```

Procédures stockées

Les procédures stockées sont des blocs de code SQL nommés et stockés dans la base de données. Elles peuvent être appelées et exécutées à partir d'une application ou d'une autre procédure stockée.

```
DELIMITER //

CREATE PROCEDURE ajouter_transaction(
    IN nom VARCHAR(100),
    IN prenom VARCHAR(100),
    IN email VARCHAR(100)
)
BEGIN
    INSERT INTO transactions (nom, prenom, email) VALUES (nom, prenom, email);
END //
DELIMITER ;
```

CALL

Pour appeler une procédure stockée, on utilise l'instruction `CALL` suivie du nom de la procédure et des paramètres nécessaires.

```
CALL ajouter_transaction('Ronaldo', 'Cristiano', 'ronaldo@gamil.fr');
```

1. Création de la Base de Données

Description : La première étape consiste à créer une base de données nommée compte_bancaire où toutes les tables et procédures seront stockées.

Code :

```sql
CREATE DATABASE compte_bancaire; -- Création de la base de données
USE compte_bancaire; -- Utilisation de la base de données
```

2. Création des Tables

Description : Ensuite, nous créons deux tables : transactions et compte. La table transactions stocke les informations sur les transactions effectuées par les clients, tandis que la table compte stocke les informations sur les comptes bancaires.

Code :

```sql
CREATE TABLE transactions (
    id INT AUTO_INCREMENT,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    PRIMARY KEY (id)
)

CREATE TABLE compte (
    id INT AUTO_INCREMENT,
    id_transaction INT,
    montant DECIMAL(10, 2),
    PRIMARY KEY (id),
    FOREIGN KEY (id_transaction) REFERENCES transactions(id)
);
```

3. Procédure de Transfert de Fonds

Description : Cette procédure permet de transférer des fonds d'un compte bancaire à un autre. Elle vérifie d'abord si le montant du compte source est suffisant pour le transfert, puis effectue la transaction en mettant à jour les montants des comptes source et destination.

Code :

```sql
DELIMITER //

CREATE PROCEDURE transfertFonds(IN id_source INT, IN id_dest INT, IN mont_transf DECIMAL(10, 2))
BEGIN
    DECLARE montant_c2 DECIMAL(10, 2);

    -- Récupérer le montant du compte source
    SELECT montant INTO montant_c2 FROM compte WHERE id_transaction = id_source;

    -- Vérifier si le montant est suffisant pour le transfert
    IF montant_c2 >= mont_transf THEN
        START TRANSACTION;

        -- Débiter le compte source
        UPDATE compte SET montant = montant - mont_transf WHERE id_transaction = id_source;

        -- Créditer le compte destination
```

```
        UPDATE compte SET montant = montant + mont_transf WHERE id_transaction = id_dest;

        COMMIT;
    ELSE
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Solde insuffisant pour effectuer le transfert.';
    END IF;
END //
DELIMITER ;
```

Utilisation : Pour transférer des fonds d'un compte à un autre, il suffit d'appeler la procédure transfertFonds en spécifiant l'ID du compte source, l'ID du compte destination et le montant à transférer.

```
CALL transfertFonds(1, 2, 500);
```

4. Procédure de Vérification de Solde

Description :

Cette procédure permet de vérifier le solde d'un compte bancaire en fonction de son ID.

Code :

```
DELIMITER //

CREATE PROCEDURE verifierSolde(IN id_compte INT)

BEGIN
    DECLARE solde DECIMAL(10, 2);

    -- Récupérer le solde du compte
    SELECT montant INTO solde FROM compte WHERE id_transaction = id_compte;

    -- Afficher le solde
    SELECT solde AS 'Solde du compte';
END //
DELIMITER ;
```

Utilisation :

Pour vérifier le solde d'un compte, il suffit d'appeler la procédure verifierSolde en spécifiant l'ID du compte.

```
CALL verifierSolde(1);
```

5. Procédure de Suppression de Compte

Description :

Cette procédure permet de supprimer un compte bancaire en fonction de son ID. Elle supprime d'abord les transactions associées au compte, puis supprime le compte lui-même.

Code :

```
DELIMITER //

CREATE PROCEDURE supprimerCompte(IN id_compte INT)

BEGIN
    START TRANSACTION;

    -- Supprimer les transactions associées au compte
    DELETE FROM transactions WHERE id = id_compte;

    -- Supprimer le compte
    DELETE FROM compte WHERE id_transaction = id_compte;

    COMMIT;
END //
DELIMITER ;
```

Utilisation :

Pour supprimer un compte bancaire, il suffit d'appeler la procédure supprimerCompte en spécifiant l'ID du compte.

```
CALL supprimerCompte(1);
```

6. procédure non générique de création des Tables users et comment et insertion des données

Description : Cette procédure crée deux tables : users pour stocker les informations des utilisateurs, et comment pour stocker les commentaires associés à chaque utilisateur.

Code :

```
DELIMITER //

CREATE PROCEDURE createTables()
BEGIN
    CREATE TABLE IF NOT EXISTS users (
        id INT AUTO_INCREMENT PRIMARY KEY,
        nom VARCHAR(100),
        prenom VARCHAR(100),
        email VARCHAR(100)
    );

    CREATE TABLE IF NOT EXISTS comment (
        id INT AUTO_INCREMENT PRIMARY KEY,
        id_user INT,
        commentaire TEXT,
        FOREIGN KEY(id_user) REFERENCES users(id)
    );
```

```
    SELECT 'Tables Users, Comment créées ou déjà existantes.' AS Resultat;
END //

DELIMITER ;
```

Utilisation :

Pour créer les tables users et comment, il suffit d'appeler la procédure createTables.

```
CALL createTables();
```

7. Procédure générique pour l'insertion des données dans une table

Description : Cette procédure permet d'insérer un utilisateur et son commentaire associé dans les tables users et comment. L'ID de l'utilisateur est automatiquement généré et utilisé pour associer le commentaire.

Code :

```
DELIMITER //

CREATE PROCEDURE insertData(IN nom VARCHAR(100), IN prenom VARCHAR(100), IN email VARCHAR(100), IN co
BEGIN
    DECLARE user_id INT;

    INSERT INTO users (nom, prenom, email) VALUES (nom, prenom, email);
    SET user_id = LAST_INSERT_ID();

    INSERT INTO comment (id_user, commentaire) VALUES (user_id, commentaire);

    SELECT CONCAT('Données insérées avec succès. ID utilisateur : ', user_id) AS Resultat;
END //

DELIMITER ;
```

Utilisation :

Pour insérer des données dans les tables users et comment, il suffit d'appeler la procédure insertData en spécifiant le nom, le prénom, l'email et le commentaire de l'utilisateur.

```
CALL insertData('Dupont', 'Jean', 'jean@gmail.com', 'Ceci est un commentaire.');
```

8. Procedure générique pour insertion Multiple de Données

Description : Cette procédure permet d'insérer plusieurs utilisateurs et leurs commentaires respectifs en une seule opération. Les utilisateurs et les commentaires sont fournis sous forme de chaînes de caractères formatées.

Code :

```
DELIMITER //

CREATE PROCEDURE insertMultipleData(
    IN user_names TEXT,
    IN commentaires TEXT
)
BEGIN
    DECLARE user_nom VARCHAR(100);
    DECLARE user_prenom VARCHAR(100);
    DECLARE user_email VARCHAR(100);
    DECLARE user_id INT;
    DECLARE i INT DEFAULT 1;
    DECLARE user_count INT;
    DECLARE comment_count INT;

    SET user_count = LENGTH(user_names) - LENGTH(REPLACE(user_names, '|', '')) + 1;
    SET comment_count = LENGTH(commentaires) - LENGTH(REPLACE(commentaires, '|', '')) + 1;

    IF user_count != comment_count THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le nombre d\'utilisateurs doit correspondre au nor
    END IF;

    WHILE i <= user_count DO
        SET user_info = SUBSTRING_INDEX(SUBSTRING_INDEX(user_names, '|', i), '|', -1);
        SET comment_info = SUBSTRING_INDEX(SUBSTRING_INDEX(commentaires, '|', i), '|', -1);

        SET user_nom = TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(user_info, ',', 1), ',', -1));
        SET user_prenom = TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(user_info, ',', 2), ',', -1));
        SET user_email = TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(user_info, ',', 3), ',', -1));

        INSERT INTO users (nom, prenom, email) VALUES (user_nom, user_prenom, user_email);
        SET user_id = LAST_INSERT_ID();

        INSERT INTO comment (id_user, commentaire) VALUES (user_id, comment_info);

        SET i = i + 1;
    END WHILE;

    SELECT 'Données insérées avec succès.' AS Resultat;
END //

DELIMITER ;
```

Utilisation :

Pour insérer des données pour plusieurs utilisateurs et leurs commentaires respectifs, il suffit d'appeler la procédure insertMultipleData en spécifiant les noms, prénoms, emails et commentaires des utilisateurs sous forme de chaînes de caractères formatées.

```
CALL insertMultipleData(
    'Dolby, Marius, dolby@gmail.com|Biden, Joe, joe@gmail.com',
    'Commentaire pour Dolby Marius|Commentaire pour Joe Biden'
```

```
);
```

9. Procédure pour Supprimer un Utilisateur et ses Commentaires

Description :

Cette procédure permet de supprimer un utilisateur et tous ses commentaires associés en fonction de son ID.

Code :

```
DELIMITER //

CREATE PROCEDURE supprimerUtilisateur(IN id_user INT)

BEGIN
    START TRANSACTION;

    DELETE FROM comment WHERE id_user = id_user;
    DELETE FROM users WHERE id = id_user;

    COMMIT;
END //

DELIMITER ;
```

Utilisation :

Pour supprimer un utilisateur et ses commentaires associés, il suffit d'appeler la procédure supprimerUtilisateur en spécifiant l'ID de l'utilisateur.

```
CALL supprimerUtilisateur(1);
```

10. Gestion des Procédures Stockées

Description :

Cette procédure permet de lister toutes les procédures stockées présentes dans la base de données.

Code :

```
SHOW PROCEDURE STATUS WHERE Db = 'compte_bancaire'; -- Afficher les procédures stockées
SET FOREIGN_KEY_CHECKS = 0; -- Désactiver les contraintes de clé étrangère
SET FOREIGN_KEY_CHECKS = 1; -- Activer les contraintes de clé étrangère
DROP PROCEDURE insertMultipleData; -- Supprimer une procédure stockée
```

VERRROUILLAGES

Les verrouillages sont des mécanismes utilisés pour contrôler l'accès concurrentiel aux données dans une base de données. Ils permettent d'éviter les problèmes de concurrence, tels que les lectures sales, les lectures non répétables et les écritures fantômes.

Types de Verrouillages

Il existe plusieurs types de verrouillages en gestion de base de données :

1. `Verrouillage en lecture` (Shared Lock) : Permet à plusieurs transactions de lire les données, mais empêche les transactions d'écrire sur les données.
2. `Verrouillage en écriture` (Exclusive Lock) : Empêche les autres transactions de lire ou d'écrire sur les données.
3. `Verrouillage en écriture partagée` (Update Lock) : Permet à une transaction de lire les données, mais empêche les autres transactions de lire ou d'écrire sur les données.

Niveaux d'Isolation

Les verrouillages sont associés à des niveaux d'isolation qui définissent le degré de séparation entre les transactions concurrentes. Les niveaux d'isolation les plus courants sont :

1. `READ UNCOMMITTED` : Permet les lectures sales, les lectures non répétables et les écritures fantômes.
2. `READ COMMITTED` : Empêche les lectures sales, mais autorise les lectures non répétables et les écritures fantômes.

Exemple de Verrouillage

```
START TRANSACTION;

SELECT * FROM compte WHERE id = 1 FOR UPDATE; -- Verrouillage en écriture
UPDATE compte SET montant = montant - 100 WHERE id = 1;

COMMIT;
```

Dans cet exemple, une transaction commence par verrouiller les données du compte avec l'ID 1 en écriture. Cela empêche les autres transactions de lire ou d'écrire sur ces données jusqu'à ce que la transaction soit validée ou annulée.

Conclusion

Les transactions et les verrouillages sont des concepts essentiels en gestion de base de données pour garantir la cohérence et l'intégrité des données. En comprenant comment les transactions fonctionnent et comment les verrouillages sont utilisés, les développeurs peuvent concevoir des systèmes de base de données robustes et fiables.