Les transactions

Fred Hémery

IUT Béthune
Département
Réseaux & Télécommunications

Base de Données (IC5) — 07/08

Introduction

- Soit une application bancaire dans laquelle on doit faire un transfert de fonds d'un compte vers un autre. Si une panne se déclare entre deux opérations, la base de données devient incohérente.
- Pour éviter ce type de problème, on va rendre invisible (atomique) les deux opérations. De cette façon, soit elle a lieu (le retrait et le crédit ont été effectués), soit elle n'a pas eu lieu (aucune des deux opérations n'a été effectué)

Définition : Une transaction est une unité de traitement séquentiel constituée d'une suite d'instructions à réaliser sur la base de données, et qui appliquée à une base cohérente, restitue une base cohérente.

État d'une transaction

- Une transaction peut se trouver dans quatre états différents :
  - **Active** : état initial conservé tant qu'aucune anomalie ne se produit ;
  - **Partiellement validée** : lorsque la dernière instruction de la transaction a été atteinte ;
  - **Échouée** : suite a une anomalie ;
  - **Validée** : après une exécution totalement terminée.
- On marque le début d'une transaction avec la clause BEGIN TRANSACTION
- L'action de validation d'une transaction est effectuée par la clause COMMIT
- Si l'exécution a échouée, le SGBD doit revenir à l'état précédent le début d'exécution (dernier point cohérent). L'action est effectuée en utilisant la clause ROLLBACK.

Les transactions avec JDBC

- Sans précision supplémentaire, toute requête SQL exécutée à partir de JDBC est validée après son exécution.
- Il est possible pour le programmeur d'indiquer "manuellement", quand il veut que la transaction soit validée.
- Le programmeur indique d'abord au pilote (driver) qu'il passe en mode "*validation manuelle*" en utilisant la méthode `setAutoCommit()` de la classe `Connection`.

```
connexion.setAutoCommit(false);
```

- Ceci est très utile lorsque l'on veut valider une transaction qui contient plusieurs instructions

Un exemple

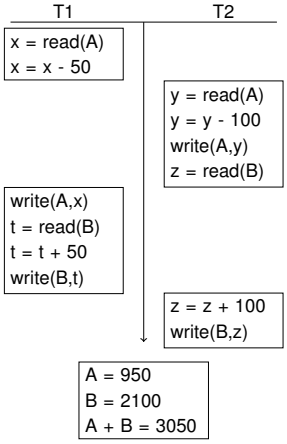
```
boolean echec = false;
try {
      connexion.serAutoCommit(false);
      PreparedStatement updateAdmission =
         connexion.preparedStatement("update admission set nb = nb + 1 where" +
                                     " type like ?");
      updateAdmission.setString(1,"fracture");
      updateAdmission.executeUpdate();
      PreparedStatement updateTotal =
         connexion.preparedStatement("update admission set total = total + 1" +
                                     " where type like ?");
      updateTotal.setString(1,"fracture");
      updateTotal.executeUpdate();
      connexion.commit();
    } catch (SQLException e ) {
      System.err.println("erreur execution requete SQL " + e.toString());
      echec = true;
    }
if (echec)
    connexion.rollback();
```


L'accès concurrent

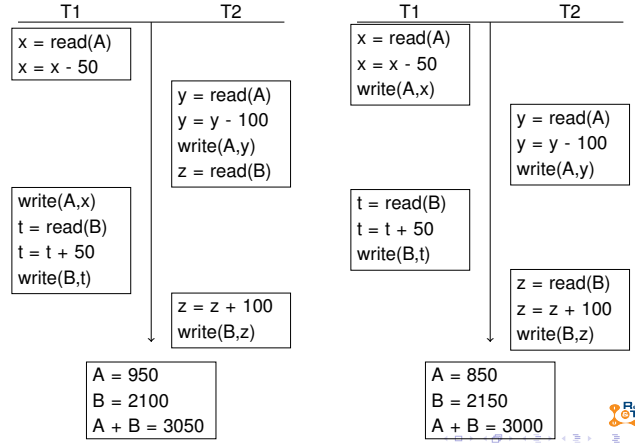
- Dans un environnement multi-utilisateurs, plusieurs opérations peuvent être effectuées en même temps sur les mêmes données. L'ordre d'exécution de ces opérations peut conduire à des incohérences dans la base de données.
- Si on reprend l'exemple du compte bancaire ; si les deux comptes sont A et B, quel que soit le montant viré de A vers B, la somme A+B doit être constante.
- Le contrôle de concurrence d'un SGBD assure que l'exécution simultanée de transactions produira le même résultat qu'une exécution séquentielle.

Définition : L'exécution d'une ensemble de transactions est dite **sérialisable** si elle donne pour chaque transaction participante, le même résultat que l'exécution en série de ces mêmes transactions.

L'accès concurrent



L'accès concurrent



Le système de verrouillage

- Pour assurer des exécutions sérialisables on utilise des verrous (il existe d'autres solutions)
- Il existe trois types de verrou
  - Les verrous posés en cas de lecture. Aucune autre transaction ne doit pouvoir modifier la donnée tant que la transaction n'est pas terminée : MODE PARTAGÉ ; lockS (*Shared*)
  - Les verrous posés en cas de selection pour mise à jour. D'autres transactions peuvent lire les données sans pouvoir les modifier : MODE PROTÉGÉ ; lockP (*Protected*)
  - Les verrous posés en cas d'écriture. Aucune autre transaction ne peut lire : MODE EXCLUSIF ; lockX (*eXclusive*)

L'approche SQL

Le standard SQL propose 4 niveaux d'isolation des transactions qui permettent d'éviter 3 types d'incohérence lorsque des transactions concurrentes s'exécutent.

- DIRTY READ : Une transaction peut lire des données qui sont modifiées par une autre transaction, sans que celle-ci est validée les modifications
- NONREPEATABLE : read Une transaction lit des données qu'elle a déjà lu et constate des modifications (une autre transaction a validé des modifications entre temps)
- PHANTOM : read Une transaction exécute une requête qu'elle a déjà exécuté et constate des modifications dans le résultat (une autre transaction a validé des modifications entre temps)

L'approche SQL

Voici les quatre niveaux

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

TAB.: Niveaux de protections

Le SQBD postgreSQL propose deux niveaux sur les quatre :

- *read committed*
- *serializable*