

Bases de données, ENS Cachan & Ulm

TP n° 5 – Procédures stockées, Transactions

Pierre Senellart (pierre@senellart.com)

4 avril 2008

Le but de ce TP est de découvrir les procédures stockées de MySQL, et les différentes fonctionnalités de gestion des transactions pour résoudre les problèmes de reprise sur panne et de concurrence, en particulier avec le moteur de stockage InnoDB.

I Interface de virement bancaire

On veut réaliser, en PHP et MySQL, une interface très simple de virement entre comptes bancaires. Un compte est simplement identifié par un entier unique, et ne peut avoir de solde négatif. L'historique des virements doit également être conservé, avec montant, comptes d'origine et de destination du virement, et date et heure de l'opération. Les virements ne peuvent bien entendu pas avoir de montant négatif.

1. Concevoir soigneusement le schéma de la base de données correspondante et créer les tables. Pour stocker un montant décimal, on pourra utiliser le type DECIMAL (voir aide en ligne). Pour stocker la date et heure d'une mise à jour MySQL, on peut utiliser une colonne de type TIMESTAMP dont la valeur par défaut vaut CURRENT_TIMESTAMP.
2. Ajouter à la main quelques comptes bancaires.
3. Écrire une page Web en PHP qui affiche la liste des comptes et leur solde, ainsi que la somme des soldes (cette somme doit *a priori* rester constante).
4. Prévoir sur cette même page un formulaire permettant d'effectuer un virement entre deux comptes (le script qui effectue ce virement n'est pas à écrire tout de suite). On pourra par exemple avoir deux listes déroulantes <select>, remplies à partir de la base de données MySQL, correspondant aux comptes d'origine et de destination.

2 Procédures stockées MySQL

MySQL permet de stocker, à l'intérieur même du système de gestion de bases de données, des procédures écrites dans un langage de programmation étendant SQL. Une procédure stockée est définie comme suit :

```
CREATE PROCEDURE nom_procédure(paramètres)
BEGIN
  déclarations

  instructions
END;
```

paramètres : liste des paramètres d'entrée et de sortie de la procédure séparés par des virgules, chacun d'entre eux de la forme « accès nom type », où accès vaut soit IN (paramètre d'entrée), soit OUT (paramètre de sortie), nom est le nom du paramètre (usuellement préfixé de p_, afin d'éviter toute confusion avec des noms d'attributs) et type est un type SQL simple (p. ex., INT ou VARCHAR(20)).

déclarations : déclarations de variables locales, terminées par des points-virgules, de la forme « DECLARE nom type; », où nom est le nom de la variable (usuellement préfixé de v_) et type est un type SQL simple.

instructions : instructions terminées par des points-virgules, dont voici les plus utilisées :

SELECT ... INTO v FROM ... ;	requête SQL renvoyant une unique ligne, stockée dans le paramètre ou variable <i>v</i>
UPDATE ... ;	ordre de mise à jour SQL
INSERT ... ;	ordre d'insertion SQL
DELETE ... ;	ordre de suppression SQL
SET v = expression ;	affectation d'une valeur au paramètre ou à la variable <i>v</i>
IF (test) THEN ... ELSE ... END IF ;	instruction de test (on peut également utiliser ELSEIF ... THEN)

Les déclarations de procédure sont généralement placées dans un fichier texte qui sera importé dans l'outil de ligne de commande mysql avec la commande `source fichier`. Cependant, quelques précautions doivent être prises :

- Il y a une confusion entre le point-virgule utilisé à l'intérieur d'une procédure stockée et celui utilisé pour conclure une commande dans l'outil mysql. Par conséquent, on redéfinit en général l'instruction de délimitation de commande avec, par exemple `DELIMITER //` (pour faire de `//` le nouveau délimiteur).
- Comme on est souvent amené à recréer une procédure, on commence souvent par supprimer la procédure existante avec un `DROP PROCEDURE nom_procédure`.

Un exemple de fichier `somme.sql` (téléchargeable depuis la page Web des TP) est ainsi :

```
DROP PROCEDURE somme_soldes_sup;

DELIMITER //
CREATE PROCEDURE somme_soldes_sup(IN p_sup DECIMAL(10,2),
                                OUT p_total DECIMAL(10,2))
BEGIN
    SELECT SUM(solde) INTO p_total FROM Comptes WHERE solde >= p_sup;
END;
//
DELIMITER ;
```

Une fois une telle procédure définie, on peut l'appeler (et récupérer le résultat) avec :

```
CALL somme_soldes_sup(100, @x); SELECT @x;
```

1. Récupérer `somme.sql`, adapter au schéma de votre base, importer dans l'outil mysql et tester.
2. Écrire une procédure stockée `virement` effectuant un virement bancaire. Cette procédure pourra mettre dans un paramètre de sortie un entier indiquant ou non le succès du virement (le virement ne pouvant pas avoir lieu quand le solde est insuffisant, quand le virement lui-même est négatif, ou quand les comptes sont inexistant).
3. Utiliser cette procédure stockée depuis PHP pour compléter l'application Web de virement. Gérer les cas où la procédure signale une erreur. On pourra utiliser l'idiome suivant pour récupérer la valeur d'un paramètre de sortie en PHP :

```
mysql_query("call virement(1,2,$i,@param)");
$resultset=mysql_query("SELECT @param");
$line=mysql_fetch_array($resultset);
$param=$line[0]
```

4. Tester le bon fonctionnement de l'application.

3 Transactions

Un virement bancaire est un exemple type d'application où il est important d'avoir des transactions : soit les différentes opérations de mises à jour sont toutes réalisées, soit aucune ne l'est. Sinon, des incohérences peuvent surgir et de l'argent peut apparaître ou disparaître sans raison !

1. Ajouter à la procédure stockée de virement, juste avant et juste après le premier ordre de mise à jour, un délai artificiel avec l'instruction `SELECT SLEEP(n) INTO v_dummy`; où `n` est le nombre de secondes à attendre (par exemple, 5), et `v_dummy` une variable entière qui aura préalablement été déclarée. Ce délai artificiel va nous permettre de simuler un délai qui serait lié à des communications réseau, une attente de confirmation d'un utilisateur, etc.
2. Effectuer un virement dans une fenêtre d'un navigateur, et, dans une autre, afficher la liste des comptes ouverts, en actualisant la page régulièrement. Que constate-t-on comme incohérence ? D'où vient-elle ?
3. Effectuer un virement dans une fenêtre d'un navigateur, et, depuis le shell, utiliser `mysqladmin` pour interrompre (`kill`) la requête en cours (voir la documentation de cet outil). Que constate-t-on comme incohérence ? D'où vient-elle ?
4. Effectuer deux virements en même temps dans une fenêtre d'un navigateur, en s'arrangeant pour faire apparaître une incohérence. D'où vient-elle ?
5. Une première solution pour résoudre ces incohérences est de poser un verrou sur une table (en écriture ou en lecture, suivant le cas) avant d'effectuer toute séquence d'opération réalisant une transaction sur cette table. Un seul verrou en écriture peut être détenu à un moment donné, et un verrou en lecture ne peut être détenu en même temps qu'un verrou en écriture. Les ordres pertinents sont :

```
LOCK TABLES table1 READ, table2 READ, ...;
LOCK TABLES table1 WRITE, table2 WRITE, ...;
UNLOCK TABLES;
```

Attention : LOCK TABLES commence par relâcher tous les verrous existants.

Ajouter des verrous aux endroits appropriés (il n'est pas toujours possible de choisir les endroits les plus appropriés : une limitation technique de MySQL empêche de poser des verrous à l'intérieur d'une procédure stockée) ; lesquels des trois problèmes signalés précédemment sont-ils résolus ?

6. Utiliser des verrous globaux est une solution peu satisfaisante, car cela peut immobiliser toute une table, et, comme on l'a vu, non suffisante. Pour faire mieux, il faut utiliser un moteur de gestion de transaction. MySQL dispose d'un tel moteur, mais celui-ci ne fonctionne qu'avec le moteur de stockage InnoDB, qui n'est pas le moteur de stockage par défaut de MySQL (celui-ci s'appelle MyISAM, et présente d'autres avantages, notamment de rapidité). Pour changer le moteur de stockage d'une table, on peut utiliser la commande :

```
ALTER TABLE nom_table ENGINE=InnoDB;
```

Les transactions fonctionnent de la manière suivante :

- le début d'une transaction est indiqué par l'ordre START TRANSACTION ;
- tout ordre de mise à jour effectué à l'intérieur d'une transaction demande un verrou sur la ou les lignes correspondantes ;
- un verrou explicite peut être demandé sur une ligne donnée en ajoutant l'indication FOR UPDATE à la fin d'un ordre SELECT ;
- les verrous sont relâchés à la fin d'une transaction, soit par un ordre COMMIT qui valide la transaction, soit par un ordre ROLLBACK qui annule l'ensemble de la transaction. Une panne ou une erreur d'exécution ont le même effet qu'un ROLLBACK.

Utiliser le système transactionnel de MySQL pour résoudre les trois incohérences constatées précédemment. Valider le bon fonctionnement.

7. Le moteur InnoDB possède en fait plusieurs modes de gestion des transactions : READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE. Ces modes peuvent être choisis, avant le début d'une transaction, avec l'ordre SET TRANSACTION ISOLATION LEVEL. Expérimenter avec ces quatre modes pour comprendre les différences. Quels avantages voyez-vous à chacun de ces modes ?