Cours MAC – Laboratoire 2

Administration des utilisateurs, droits et transactions

Etape 2 – Mode transactionnel – 3 séances

ELS- 31 Octobre 2016

1 Objectif

- Etudier les différents modes transactionnels inhérents au moteur InnoDB.
- Mise en évidence des problèmes d'interblocage.
- Prévenir l'interblocage dans un cas précis.

2 Théorie – Les modes transactionnels InnoDB

La norme SQL ANSI (SQL92) définit quatre modes d'isolation correspondant à quatre compromis différents entre le degré de concurrence et le niveau d'interblocage des transactions. Ces modes d'isolation sont définis par rapport aux **5 types d'anomalies** que nous avons rencontrés en théorie dans la section « Transactions »:

- ***Perte de mise à jour – Pb no 1 & 2 (« Lost Update - Pb 1 & Pb2»)***

  Pb1 : Ecriture écrasée du fait de la plantée d'une transaction parallèle

  Pb2 : Ecriture écrasée par une transaction parallèle
- ***Lectures sales (« Dirty reads »)***

  Lecture d'une valeur intermédiaire, n'appartenant à aucun état cohérent de la BD

- ***Lectures non répétables («Non-repeatable reads »)***

  La lecture du même objet donne des valeurs différentes en raison d'une modification opérée par une transaction concurrente qui a été validée.

- ***Tuples fantômes (« Phantoms tuples »)***

  La lecture d'une même liste d'objets fait apparaitre des objets supplémentaires en raison de créations opérées par une transaction concurrente validée.

Voici les 4 modes en question, les 4 « Niveaux d'isolation »

Niveau d'isolation	Perte écriture – Pb1	Lectures sales	Lectures non répétables	Perte écriture – Pb2	Fantômes
Read Uncommitted	-	Possible	Possible	Possible	Possible
Read Committed	-	-	Possible	Possible	Possible
Repeated Read	-	-	-	- Possible avec MySQL☹	Impossible avec MySQL
Serializable	-	-	-	-	-

Garantie de mise en œuvre du mode d'isolation

Le mode mode d'isolation est garanti dans la mesure on l'on travaille en mode transactionnel, c'est à dire en encadrant les blocs transactionnels par :

```
start transaction;
  Instructions du bloc transactionnel
commit ;
```

Mode par défaut

Il existe un mode d'isolation par défaut qui varie d'un système à l'autre, le plus courant semblant être **`READ COMMITTED`**, c'est le cas notamment pour Oracle et SQLServer.

MySQL travaille par défaut dans le mode **`REPEATED READ`.**

Comme présenté dans le tableau ci-dessus, MySQL diffère quelque peu du standard REPEATED READ: Les pertes d'écritures (pb no2) sont possibles; en revanche, l'apparition de tuples fantômes n'arrive pas.

Principe de fonctionnement de ces 4 modes

La mise en œuvre de ces 4 niveaux d'isolation consiste en gros à opérer automatiquement des verrouillages en lecture et en écriture, en mode shared ou exclusif selon les cas, dès qu'une opération de lecture ou qu'une opération d'écriture est rencontrée. Puis de relâcher ces verrous à la fin de la transaction (qu'elle réussisse ou non).

Voici très grossièrement comment les SGBD mettent en œuvre ces 4 modes du point de vue des verrouillages effectuées (ou `X` représente un verrou exclusif et `S` un verrou partagé):

Niveau d'isolation	Write Lock	Read Lock	Range Lock
Read Uncommitted	X	-	-
Read Committed	X	S	-
Repeated Read	X	X	-
Serializable	X	X	X

C'est en fait un peu plus compliqué en réalité, chaque SGBD procède selon une technique qui lui est personnelle.

Spécification du niveau d'isolation

La spécification du niveau d'isolation est réalisée en saisissant une instruction MySQL :

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL level

level:REPEATABLE READ
    | READ COMMITTED
    | READ UNCOMMITTED
    | SERIALIZABLE
```

Par exemple :

```
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

Mode Autocommit

Enlever le mode autocommit (mode par défaut de MySQL)

```
set @@autocommit=0
```

Connaître le mode courant d'isolation

En lisant la valeur de la variable système `@@tx_isolation`

Comme par exemple : `SELECT @@tx_isolation`

3 A faire dans cette deuxième étape

Reprendre la base de données `transactions` mise en place dans la première étape.

Principe

Vous allez tester ces différents niveaux d'isolation en exécutant de manière simultanée des transferts entre deux comptes 1 et 2.

Chaque transfert correspondra à une transaction dont le code sera placé dans une procédure stockée.

Cette procédure stockée sera invoquée simultanément par deux threads Java :

- Le premier, travaillant avec une connexion de l'utilisateur U1, effectuera en boucle - par exemple 2000 fois - des transferts du compte 1 vers le compte 2.
- Le deuxième, travaillant avec une connexion de l'utilisateur U2, effectuera en boucle – le même nombre de fois que le premier thread - des transferts du compte 2 vers le compte 1.

Au final, les deux comptes devraient se retrouver avec un solde identique à celui qu'ils avaient au départ.

Créer une nouvelle procédure stockée « transférer »

Cette procédure sera rédigée comme ci-dessous (présentée ici en pseudo-code). Vous pourrez constater qu'il s'agit d'une rédaction très lourde, qui a comme seul intérêt de générer facilement des conflits transactionnels.

```
procedure (cpt1 varchar(30), cpt2 varchar(30), montant float)
begin
      déclarer une variable « etat »
      lire le solde du compte cpt1 dans la variable etat
      etat <- etat – montant
      mettre à jour le nouveau solde du compte cpt1 avec la valeur de etat

      lire le solde du compte cpt2 dans la variable etat
      etat <- etat + montant
      mettre à jour le nouveau solde du compte cpt2 avec la valeur de etat
end
```

Prévoir 4 versions de la procédure (4 procédures stockées !!)

`transferer1` : la procédure présentée ci-dessus, en omettant l'encadrement « start transaction commit »

`transferer2` : idem que `transférer 1`, mais travaillant en mode transactionnel, à savoir avec un encadrement « start transaction commit », mais sans opérer aucun verrouillage explicite.

`transferer3` : Idem que `transférer2`, en travaillant en mode transactionnel, mais en opérant par vos soins un verrouillage explicite des données sensibles en obéissant au « verrouillage en deux phases », en verrouillant le plus tard possible.

`transferer4` : Idem que `transférer3`, mais en opérant cette fois-ci un verrouillage des données sensibles en prévenant tout risque d'interblocage. Utiliser la méthode d'ordonnancement vue en cours : « Ordonner les articles de la base et imposer que les transactions verrouillent les articles selon cet ordre préétabli ».

Programme Java multithreadé

Ecrire un programme Java qui comportera une classe `TransfertMultiple`, dont l'instanciation donnera un **objet actif** responsable d'invoquer x fois l'une ou l'autre des procédure `transferer`.

 En cas d'interblocage, `exception` levée de type `SQLException`, avec `exception.getSQLState().equals("40001")`, rejouer la transaction !

L'instanciation d'un tel objet ressemblera à:

```
TransfertMultiple unTransfertMultiple = new TransfertMultiple ("U1") ;
```

`"U1"` → nom d'un utilisateur inscrit dans le SGBD

Pour démarrer le transfert multiple, quelque chose ressemblant à :

```
unTransfertMultiple.demarrer (
      "cpt_a",          → spécification du premier compte (dont le montant est retiré)
      "cpt_b",          → spécification du deuxième compte (dont le montant est rajouté)
      50,               → montant transféré d'un compte à l'autre
      2000,             → nombre de transferts (itérations)
      "transferer1"     → nom de la procédure de transfert à invoquer
) ;
```

Partie expérimentale

Ci-dessous sont décrits les différents tests à opérer.

 Dans tous les cas de figure, **vous décrirez** les résultats obtenus, **commenterez** et **justifierez** ces résultats : Y a-t-il eu conflit ou pas? Y a-t-il eu interblocage ou pas, et pourquoi..

 Comparez également **les temps d'exécution** (heure système) et le **nombre observé d'interblocages**.

Pour chacun des tests à opérer, la procédure sera identique :

Créer deux objets actifs :

```
TransfertMultiple transfertMultiple1 = new TransfertMultiple ("U1") ;
TransfertMultiple transfertMultiple2 = new TransfertMultiple ("U2") ;
```

Puis démarrez les transferts multiples (exemple avec un montant de 50.- transféré 2000 fois) :

```
transfertMultiple1.demarrer ("cpt_1", "cpt_2", 50, 2000,
"nom_proc_transfert" ) ;
transfertMultiple2.demarrer ("cpt_2", "cpt_1", 50, 2000,
"nom_proc_transfert" ) ;
```

Tests à opérer ➔

1. Tester le mode non transactionnel avec `transferer1`

2. Tester le mode transactionnel avec `transferer2`, `transferer3` puis `transferer4`.

   Pour chacune des trois procédures de transfert, essayer avec les 4 modes d'isolation **`Read Uncommitted`**, **`Read Committed`**, **`Repeated Read`** et **`Serializable`**

4 A rendre dans le rapport

- Présenter les codes développés en expliquant/commentant leur structure et leurs particularités
- Partie "expérimentale" : répondre à ce qui est demandé dans l'énoncé du laboratoire.

 Deux remarques importantes

1. Utiliser les 4 modes transactionnels pour les procédures transférer 2, 3 et 4, **faire un tableau** et comparer les résultats (inter-blocages ou non, cohérence, temps d'exécution).
2. La **cohérence** signifie que l'addition des soldes des deux comptes donne le même montant avant et après avoir exécuté le programme Java. Autrement dit, qu'il n'y a pas eu d'argent ni volatilisé ni créé.