POLYTECHNIQUE MONTRÉAL

INF8480 - Systèmes répartis et infonuagique

TP5 - Validateur de transactions

Chargés de laboratoire :
Daniel Capelo Borges daniel.capelo@polymtl.ca
Pierre-Frederick Denys pierre-frederick.denys@polymtl.ca

Automne 2019 - V3.0

1 Introduction

1.1 Prérequis

— **Cohérence et réplication pour les données réparties :** Transactions et procédures de recouvrement. Contrôle des opérations simultanées (verrous, méthodes optimistes, ordonnancement par identificateur de temps). Protocoles pour les mises à jour atomiques réparties. Contrôle des accès simultanés répartis. Transactions en présence de réplication.

1.2 But du TP

— Transactions sur une base de données
— Accès en lecture et en écriture conccurente sur une base de données
— Sérialisation et ordonnancement de transactions

2 Partie 1 : Transactions SQL

2.1 Introduction

Le but de cette première partie est de comprendre le principe de transaction dans une base de données. Et les enjeux des lectures et écritures conccurentes dans une base de donnée.

Par défaut, MySQL ne travaille pas avec les transactions. Les requêtes sont donc automatiquement validées et il n' y a pas de retour en arrière possible. Seules les tables InnoDB sont transactionnelles pour MySQL.

Dans cette première partie, vous allez utiliser MySQL en désactivant l'**autocommit**. Tant que vos modifications ne sont pas validées par un **commit**, vous pouvez les annuler à tout moment(**rollback**).

2.2 Mise en place d'une base de données

Vous allez travailler sur une machine virtuelle openstack. Vous pouvez réutiliser celle du dernier TP, ou suivre le TP précédent (TP3) afin de la mettre en place. Docker doit y être installé.
Vous devez lancer plusieurs containers afin de mettre en place l'architecture suivante :

— **1 serveur MySQL :**
  — A partir de l'image `ubuntu` (penser à ouvrir le port correspondant à la base MySQL), lancer un container avec le nom `server`, et s'y connecter.
  — **Sur le container** installer le paquet `mysql-server`, et autoriser l'accès externe (voir le lien `https://support.rackspace.com/how-to/installing-mysql-server-on-ubuntu/`).
  — Se connecter en tant qu'admin, et créer un utilisateur `usertp` avec le mot de passe `usertp`. Y lancer les commandes suivantes pour créer la base et les tables.

```
CREATE DATABASE outils;
```

```
CREATE TABLE tournevis (
  id INT NOT NULL,
  name VARCHAR(255),
  description VARCHAR(255),
  price VARCHAR(255)
) ENGINE = InnoDB;

CREATE TABLE vis (
  id INT NOT NULL,
  name VARCHAR(255),
  description VARCHAR(255),
  price VARCHAR(255)
) ENGINE = InnoDB;
INSERT INTO tournevis (id, name, description, price) VALUES
   (2, 'tournevis', 'carre',15);

INSERT INTO tournevis (id, name, description, price) VALUES
   (3, 'tournevis', 'etoile',19);
```

— A partir de l'image `ubuntu`, lancer deux containers `client1` et `client2` avec version cliente de MySQL pour y effectuer des requêtes. Connectez vous aux deux containers dans deux consoles séparées, et y lancer les commandes suivantes pour vous connecter à la base que vous avez créer sur le serveur. (Penser à remplacer l'adresse IP du container server.

```
mysql -h <ip_container_server> -u usertp -p
use outils
```

2.3 Annulation de transaction

Effectuer la requête sur un des clients

```
SET autocommit = 0;
INSERT INTO tournevis (id, name, description, price) VALUES (6,
   'tournevis', 'cruciforme',12);
INSERT INTO tournevis (id, name, description, price) VALUES (7,
   'tournevis', 'plat',11);
UPDATE tournevis SET price = 20 WHERE id = 3;
SELECT * FROM tournevis;
ROLLBACK;
SELECT * FROM tournevis;
```

Observer la sortie de la seconde requête.

2.4 Interruption de transaction

Effectuer la requête sur un des clients

```
INSERT INTO tournevis (id, name, description, price) VALUES (6,
   'SC001', 'cruciforme',12);
INSERT INTO tournevis (id, name, description, price) VALUES (6,
   'SC002', 'plat',11);
UPDATE tournevis SET price = 20 WHERE id = 3;
SELECT * FROM tournevis;
exit -- pour simuler une interruption
-- Se reconnecter au serveur
SET autocommit = 0;
SELECT * FROM tournevis;
```

Observer la sortie de la seconde requête.

2.5 Verrous de table

Exécuter les commandes suivantes sur un des clients :

— Pour verrouiller une table, il faut utiliser la commande `LOCK TABLES`.
  — En utilisant `READ`, un verrou de lecture sera posé ; c'est-à-dire que les autres sessions pourront toujours lire les données des tables verrouillées, mais ne pourront plus les modifier.
  — En utilisant `WRITE`, un verrou d'écriture sera posé. Les autres sessions ne pourront plus ni lire ni modifier les données des tables verrouillées.
— Pour déverrouiller les tables, on utilise `UNLOCK TABLES`

2.5.1 Expériences

```
LOCK TABLES tournevis READ, vis AS screw WRITE;
SELECT id, name FROM tournevis;
SELECT id, name FROM tournevis AS screwdriver;
```

Pourquoi la requête ne fonctionne pas ?

```
UPDATE tournevis SET price = 16 WHERE id = 2;
```

Pourquoi la requête ne fonctionne pas ?

```
SELECT id, name FROM screw;
```

Pourquoi la requête ne fonctionne pas ?
Il faut donc penser à acquérir tous les verrous nécessaires aux requêtes à exécuter.

```
UNLOCK TABLES; -- On rel che d'abord les deux verrous
   pr c dents

LOCK TABLES vis READ;
LOCK TABLES tournevis READ, tournevis AS screwdriver READ;

SELECT id, name FROM tournevis;
```

```
SELECT id, name FROM tournevis AS screwdriver;

SELECT * FROM vis;
```

Pourquoi la requête ne fonctionne pas ?

2.5.2 Accès concurent

Sélection sur des tables verrouillées à partir d'une autre session. Ouvrez deux consoles (session1 et client2) sur la même base avec le même compte.

```
LOCK TABLES tournevis READ, vis WRITE; --sur client1

SELECT id, name FROM tournevis; --sur client2

SELECT id, name FROM vis ; --sur client2
```

Que faut-il faire pour que la dernière requête s'execute correctement ?

Modification sur des tables verrouillées à partir d'une autre session. exécuter sur un des container client

```
UPDATE tournevis SET price = '14' WHERE id = 2; --sur client1
UNLOCK TABLES --sur client1
LOCK TABLES tournevis READ, vis WRITE; --sur client1
```

Verrou posé par une requête d'insertion exécuter sur un des container client

```
     START TRANSACTION;
INSERT INTO tournevis
VALUES (12, 'tournevis', 'hexagonal', 20); -- sur client1
SELECT * FROM tournevis
WHERE id > 13
LOCK IN SHARE MODE;  -- sur client1

SELECT * FROM tournevis
WHERE id < 13
LOCK IN SHARE MODE;  -- sur client1
```

3 Partie 2 : Ordonnancement de transactions

Considérez les ordonnancements de transactions suivant, et répondre aux questions sur le quiz moodle.

3.1 Cas 1

Temps	Transaction T_1	Transaction T_2	Transaction T_3	État BD
t_1	Début Transaction			A=5; B=10;
t_2	Lire(A,a);			
t_3	a:=(a*2)-5;			
t_4	Écrire(a,A);			A=5;
t_5			Début Transaction	
t_6			Lire(A,a);	
t_7			Lire(B,b);	
t_8		Début Transaction		
t_9		Lire(A,a);		
t_{10}		Lire(B,b);		
t_{11}		Afficher(a,b);		**Console:** A=5; B=10;
t_{12}		Confirmer		
t_{13}	Lire(B,b);			
t_{14}	b:=(b*2)-10;			
t_{15}	Écrire(b,B);			B=10;
t_{16}	Confirmer			
t_{17}			tmp:= a;	
t_{18}			a:= b;	
t_{19}			b:= tmp;	
t_{20}			Écrire(A,a);	A=10;
t_{21}			Écrire(B,b);	B=5;
t_{22}			Confirmer	

— Cet ordonnancement est-il sérialisable ? Si oui, donnez le ou les ordonnancements séquentiels équivalents.
— Cet ordonnancement est-il sérialisable par permutation ? Justifiez votre réponse.
— Expliquez la convergence ou la divergence des résultats des tests de sérialisabilité effectués en a) et b). Aurait-on eu le même résultat pour d'autres opérations en t_3 et t_{14} ?

3.2 Cas 2

Temps	Transaction T_1	Transaction T_2	État BD	Journal
t_1		Début Transaction	A=10; B=20;	
t_2				(*Début*, 2)
t_3		Lire(B,b);		
t_4	Début Transaction			
t_5				(*Début*, 1)
t_6	Lire(A,a);			
t_7	a := a+50;			
t_8	Écrire(a,A);			
t_9				(*Défaire*, 1, A:10)
t_{10}		b := b+20;	A=60;	
t_{11}				(Refaire, 1, A:60)
t_{12}	Lire(B,b);			
t_{13}		Lire(A,a);		
t_{14}		Écrire(b,B);		
t_{15}				(*Défaire*, 2, B:20)
t_{16}				(*Refaire*, 2, B:40)
t_{17}	Afficher A+B;		Affiche 80;	
t_{18}	ConfirmerTransaction		B=40;	
t_{19}				(*Confirmer*, 1)
t_{20}		a := a+10;		
t_{21}		Écrire(a,A);		
t_{22}				(*Défaire*, 2, A:60)
t_{23}		ConfirmerTransaction	A=70;	
t_{24}				(*Refaire*, 2, A:70)
t_{25}				(*Confirmer*, 2)

— Cet ordonnancement est-il sérialisable par permutation ? Justifiez votre réponse. Si l'ordonnancement est sérialisable, donnez un ordonnancement séquentiel équivalent.

— Cet ordonnancement satisfait-il le protocole de contrôle de concurrence par estampillage (timestamping) ? Justifiez votre réponse. Dans le cas où l'ordonnancement ne satisfait pas le protocole, dites quelle(s) transaction(s) sera annulée(s).

— Supposons qu'une panne survient au temps t_{24}. Dites quels enregistrements du journal seront utilisés pour la récupération si l'algorithme en une passe est employé. Donnez l'état de la BD après la récupération.

3.3 Cas 3

Temps	Transaction T_1	Transaction T_2	État BD
t_1	Début Transaction		A=60; B=40
t_2	Ver(A,P)		
t_3	Lire(A,a)		
t_4	Dev(A)		
t_5		DébutTransaction	
t_6		Ver(A,P)	
t_7		Lire(A,a)	
t_8		Dev(A)	
t_9	Ver(B,X)		
t_{10}	b:=a		
t_{11}	Écrire(b,B)		B=60
t_{12}	Dev(B)		
t_{13}		Ver(B,X)	
t_{14}		b:=a+5	
t_{15}		Écrire(B,b)	B=65
t_{16}		Dev(B)	
t_{17}	Ver(A,X)		
t_{18}	a:= a / 2		
t_{19}	Écrire(a,A)		A=30
t_{20}	Dev(A)		
t_{21}	ConfirmerTransaction		
t_{22}		ConfirmerTransaction	A=30; B=65

— Est-ce que cet ordonnancement est sérialisable ?
— Est-ce que cet ordonnancement est sérialisable par permutations ? Jutifiez votre réponse.
— Cet ordonnancement respecte-t-il le protocole de verrouillage en 2 phases (V2P) ?

♢