Concurrence - Travaux Pratiques

Ce TP de concurrence devra être réalisé sous la base de données **MySQL** avec le moteur de stockage InnoDB (le seul qui soit capable de supporter tous les niveaux d'isolation).

Nous utiliserons deux consoles, dans lesquelles nous exécuterons successivement des commandes correspondant aux opérations respectives de deux transactions. Cela nous permettra de mettre en évidence les anomalies liées à un contrôle de concurrence trop lâche, avec des défauts d'isolation entre les transactions "à gauche" et "à droite". Nous verrons que les niveaux d'isolation du standard SQL permettent de se prémunir contre certains de ces défauts.

1. But de la séances de TP

Dans ce TP, vous devrez trouver pour chaquer niveau d'isolation, un enchainement d'opérations (une histoire) dans 2 sessions différentes (deux terminaux) entre deux transactions[1], pour mettre en valeur le ou les défauts d'isolation proposés.

1. Mode `READ UNCOMMITTED` : Lecture sale
2. Mode `READ COMMITTED`[2] : Écriture sale, tuple fantôme
3. Mode `REPEATABLE READ` : Tuple fantôme [3]
4. Mode `SERIALISABLE`[4] : *Dead lock*

2. Instructions

1. Sous l'environnement *WampServer*, il v avec la console (bouton gauche sur l'icone de WampServer, puis 'MySQL', enfin 'Console'), ou mysql sous linux.
2. Pour pouvoir constater des problèmes de concurrences, il vous faut deux sessions différentes. Il vous faudra donc ouvrir 2 terminaux sous mysql. Pour les différencier, nous utiliserons la commande :
   'PROMPT SESSION1> ' (resp SESSION2).
3. Pour chaque session, il faut enlever le mode de validation automatique :
   `SET AUTOCOMMIT = 0;`
4. Lorsque vous devrez changer de mode d'isolation :
   - `SET SESSION TRANSACTION ISOLATION LEVEL` **READ UNCOMMITTED**;

[1] Ne pas oublier de supprimer le mode auto-commit et ni de vérifier le niveau d'isolation dans CHAQUE session

[2] Sous MySQL, la lecture sale est résolue grâce au *versioning*, qui permet de garder l'image de la donnée avant la transaction

[3] Sous MySQL, les lectures fantômes, théoriquement possibles dans le standard SQL, ne sont pas possibles. En revanche, les écritures fantômes sont possibles. C'est-à-dire, mettre à jour des lignes qui n'étaient pas là au début d'une transaction.

[4] Comme MySQL est en versionning, le deadlock ne peut arriver que sur une histoire de la sorte $w_1(x)w_2(y)w_1(y)w_2(x)$

- SET SESSION TRANSACTION ISOLATION LEVEL **READ COMMITTED**;
- SET SESSION TRANSACTION ISOLATION LEVEL **REPEATABLE READ**;
- SET SESSION TRANSACTION ISOLATION LEVEL **SERIALIZABLE**;

3. Schéma de données et Données

Schéma Pour tester la concurrence entre deux transactions, nous allons utiliser les trois tables suivantes :

```
USE TEST; -- Sélectionne la base de données
DROP TABLE Spectacle; -- supprime les tables si elles existaient déjà
DROP TABLE Client;
DROP TABLE Reservation;

CREATE TABLE Spectacle (id_spectacle INT NOT NULL PRIMARY KEY,
                        places_offertes INT NOT NULL,
                        places_libres INT NOT NULL,
                        tarif DECIMAL(10,2) NOT NULL
            ) ENGINE=InnoDB;

CREATE TABLE Client (id_client INT NOT NULL PRIMARY KEY,
                     Solde FLOAT NOT NULL
            ) ENGINE=InnoDB;

CREATE TABLE Reservation (id_client INT NOT NULL,
                          id_spectacle INT NOT NULL,
                          places_reservees INT NOT NULL,
                          PRIMARY KEY (id_client, id_spectacle),
                          KEY spectacle (id_spectacle)
            ) ENGINE=InnoDB;
```

Pour que la base de données reste cohérente, il faut que pour un spectacle donné : $sum(places\_reservees) = (places\_offertes - places\_libres)$ (tables Réservation et Spectacle, respectivement).

Données L'état d'origine de notre base de données est donné par les requêtes suivantes :

```
DELETE FROM Reservation;
DELETE FROM Client;
DELETE FROM Spectacle;
INSERT INTO Client VALUES (1, 50);
INSERT INTO Client VALUES (2, 50);
INSERT INTO Spectacle VALUES (1, 250, 250, 20);
COMMIT;
SET SESSION TRANSACTION ISOLATION LEVEL ... ;
```

Entre chaque test de concurrence sur notre schéma, la base de données doit avoir cet état pour être cohérent. Ainsi, vous pourrez mettre à zéro la base en exécutant ce script.

4. Transactions

Voici le jeu de transactions qui vous permettra de mettre en valeur les erreurs ou cas demandés (section 1.).

Attention ! Une **séquence de requêtes n'est pas interchangeable**. Les séquences ne sont pas intégrées dans des procédures, pour permettre de décomposer les opérations dans le temps et favoriser la concurrence. Vous devrez donc vérifier vous-même les conditions (T_1 et T_2) avec les valeurs LOCALES '@' à la session, et le cas échéant, faire un ROLLBACK et arrêter la transaction.

Pour faciliter la découverte d'erreur, nous vous invitons à modéliser chaque transaction sous forme de séquences d'opérations :

1. Donner pour chaque transaction les séquences d'opérations possibles. Noter, le cas échéant, les conditions d'application de cette transaction (exemple : @nb_places < 2) ;

2. Présenter par la suite, chaque histoire sous forme de suite d'opérations grâce aux transactions que vous venez de produire[5].

T_1 Réservation de 2 places pour le client 1

$R_1(sp_1)$	SELECT places_libres, tarif INTO @nb_libres, @tarif FROM Spectacle WHERE id_spectacle = 1 ;
r_1	*Vérifier si "SELECT @nb_libres - 2 ;" < 0 alors ROLLBACK ;*
$W_1(sp_1)$	UPDATE Spectacle SET places_libres = @nb_libres - 2 WHERE id_spectacle = 1 ;
$W_1(re_1)$	INSERT INTO Reservation VALUES (1, 1, 2) ;
$R_1(so_1)$	SELECT Solde INTO @solde FROM Client WHERE id_client = 1 ;
r_1	*Vérifier si "SELECT @solde - 2 * @tarif ;" < 0 alors ROLLBACK ;*
$W_1(cl_1)$	UPDATE Client SET Solde = @solde - 2 * @tarif WHERE id_client = 1 ;
c_1	COMMIT ;

T_2 Réservation de 5 places pour le client 2

$R_2(sp_1)$	SELECT places_libres, tarif INTO @nb_libres, @tarif FROM Spectacle WHERE id_spectacle = 1 ;
r_2	*Vérifier si "SELECT @nb_libres - 5 ;" < 0 alors ROLLBACK ;*
$W_2(sp_1)$	UPDATE Spectacle SET places_libres = @nb_libres - 5 WHERE id_spectacle = 1 ;
$W_2(re_2)$	INSERT INTO Reservation VALUES (2, 1, 5) ;
$R_2(so_2)$	SELECT Solde INTO @solde FROM Client WHERE id_client = 2 ;
r_1	*Vérifier si "SELECT @solde - 5 * @tarif ;" < 0 alors ROLLBACK ;*
$W_2(cl_2)$	UPDATE Client SET Solde = @solde - 5 * @tarif WHERE id_client = 2 ;
c_2	COMMIT ;

T_3 Vérification du nombre de places réservées

$R_3(re_{1,2})$	SELECT SUM(places_reservees) AS places_reservation FROM Reservation WHERE id_spectacle = 1 ;
$R_3(sp_1)$	SELECT (places_offertes - places_libres) AS places_spectacle FROM Spectacle WHERE id_spectacle = 1 ;
c_3	COMMIT ;

[5] Tester sous MySQL pour vous assurer du bon fonctionnement

T_4 **Mise à jour des places pour spectacle et client**

$W_4(sp_1)$ UPDATE Reservation SET places_reservees = places_reservees + 10
WHERE id_client = 2 AND id_spectacle = 1 ;

$W_4(cl_1)$ UPDATE Spectacle SET places_offertes = places_offertes + 10, places_libres = places_libres + 10
WHERE id_spectacle = 1 ;

c_4 COMMIT ;