

TP n° 2

Consignes les exercices ou questions marqués d'un * devront être d'abord rédigés sur papier (afin de se préparer aux épreuves écrites de l'examen). En particulier, il est recommandé d'être dans les mêmes conditions qu'en examen : pas de document ni de calculatrice. Tous les TPs se font sous Linux.

1 Sérialisabilité *

Pour les deux transactions données ci-dessous, dire si les ordonnancements proposés sont sérialisables (justifier) :

(a)

T_1	T_2
$r_{T_1}(X)$	
$r_{T_1}(Y)$	
$w_{T_1}(X)$	$w_{T_2}(X)$
	$w_{T_2}(Y)$

(b)

T_1	T_2	T_3
$r_{T_1}(X)$		$r_{T_3}(Y)$
$w_{T_1}(X)$	$w_{T_2}(Y)$	
	$w_{T_2}(X)$	$r_{T_3}(X)$

2 Verrous *

Proposer deux transactions SQL simples provoquant un *deadlock*.

3 JDBC

Récupérer l'archive du TP2 et l'importer dans Eclipse. Le but de cet exercice est d'utiliser des threads dans un programme Java pour établir deux connexions simultanées à une base et effectuer des mises à jour concurrentes tout en s'assurant qu'un certain invariant de la table est préservé.

- compléter la méthode `init()` de la classe `TestDB`. La méthode doit récupérer une connexion à la base et s'en servir pour :
 - détruire la table `T` si elle existe
 - créer la table `T` ayant trois attributs entiers `id`, `x` et `y`
 - Remplir la table avec des valeurs $(i, i, 0)$ pour i entre 0 et 999 inclus.

Lancer le programme Java après avoir modifié la méthode `main()` en y mettant les bons identifiants de connexion à la base. Se connecter à la base en console avec `psql` et constater que la table est bien remplie.

- La méthode `complex(int x)` (qui renvoie la constante 42) joue ici le rôle d'un fragment de code complexe, définit en Java et exprimant une partie de la logique de l'application (par exemple ça pourrait être un traitement complexe utilisant des fichiers présents sur le poste client). On souhaite effectuer les mises à jour suivante :

traitement 1 trouver toutes les lignes de `T` où `x` est pair et mettre à jour la case `y` correspondante par `complex(y)` (i.e. 42).

traitement 2 de manière concurrente, rechercher tout les lignes où `x` est pair et `y` vaut 0 et faire `x = x + 1` (c'est à dire rendre `x` impair)

On souhaite préserver l'invariant suivant : **les lignes pour lesquelles `x` est impair ont un `y` valant 0**.
On procède en plusieurs étapes.

- (a) Dans la méthode `concurrentUpdate()`, créer un objet `Thread` dont la méthode `run` envoie un ordre SQL effectuant le traitement 2 ci-dessus (la fonction `MOD(a,b)` en SQL permet de calculer le reste de a/b).
- (b) Compléter la suite de la méthode `concurrentUpdate()` pour effectuer le traitement 1 en deux temps. D'abord trouver tous les candidats au moyen d'un `SELECT`, puis parcourir le résultat en Java et mettre à jour ligne à ligne.

décommenter l'appel de méthode dans le `main` et constater que l'invariant n'est pas respecté (dans la console `psql`)

3. (bonus) rajouter une contrainte au schéma de la table `T` pour que l'invariant soit vérifié lors des modifications de la table et constater que votre programme lève maintenant une erreur.
4. Corriger votre programme pour respecter l'invariant.