

## 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 Sériabilité \*

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

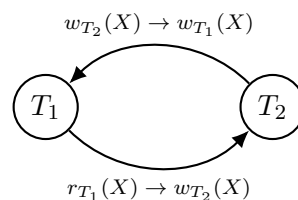
(a)

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

(b)

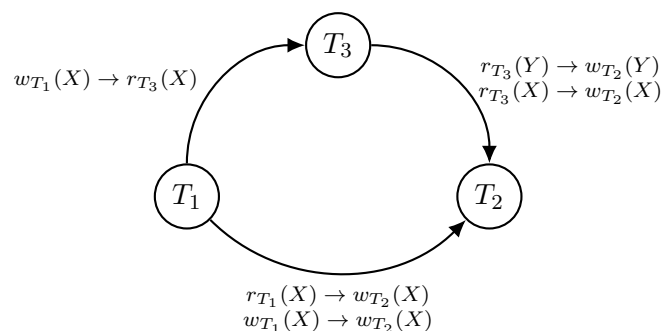
$T_1$	$T_2$	$T_3$
$r_{T_1}(X)$		
$w_{T_1}(X)$		$r_{T_3}(Y)$
	$w_{T_2}(Y)$	
	$w_{T_2}(X)$	$r_{T_3}(X)$

**Réponse:** (a) L'ordonnement n'est pas sérialisable. Il suffit de tracer le graphe dont les sommets sont  $T_1$  et  $T_2$  et de rajouter un arc entre  $T_1$  et  $T_2$  s'il existe deux actions en conflit dont l'une est faite en  $T_1$  et l'autre faite en  $T_2$  après la première (et *vice versa*).



On se rend compte qu'on a un arc  $T_1 \rightarrow T_2$  à cause de la paire d'action  $(r_{T_1}(X), w_{T_2}(X))$  et un arc  $T_2 \rightarrow T_1$  à cause de la paire d'action  $(w_{T_2}(X), w_{T_1}(X))$ . On peut s'arrêter là car un seul conflit suffit à montrer que l'ordonnement n'est pas sérialisable.

(b) L'ordonnement est sérialisable :



Si on trace le graphe complet, on se rend compte qu'il n'a pas de cycle et que l'ordonnement est équivalent à exécuter d'abord  $T_1$  puis  $T_3$  et enfin  $T_2$ .

## 2 Verrous ★

Proposer deux transactions SQL simples provoquant un *deadlock*.

**Réponse:** On suppose une table  $T(id\text{INTEGER}, bal\text{INTEGER})$  contenant les tuples (1, 1000) et (2, 1000). Les transactions suivantes, provoquent un *deadlock* en PostgreSQL en utilisant le mode d'isolation par défaut :

```
BEGIN | BEGIN
UPDATE T SET bal = bal + 100 WHERE id = 1 |
| UPDATE T SET bal = bal + 100 WHERE id = 2
UPDATE T SET bal = bal - 100 WHERE id = 2 |
| UPDATE T SET bal = bal - 100 WHERE id = 1
```

La première action (à gauche) prend le verrou sur la ligne d'id 1 en lecture et en écriture.

La deuxième action (à droite) prend le verrou sur la ligne d'id 2 en lecture et en écriture.

La troisième action (à gauche) veut prendre le verrou sur la ligne d'id 2 en lecture et en écriture. Elle est bloquée car le verrou n'a pas été relâché par la transaction de droite.

La quatrième action (à droite) veut prendre le verrou sur la ligne d'id 1 en lecture et en écriture. Elle est bloquée car le verrou n'a pas été relâché par la transaction de gauche, on obtient un *deadlock* (un cycle dans les attentes de verrou) détecté par PostgreSQL. La deuxième transaction est alors interrompue en erreur (et elle relâche le verrou 2) ce qui permet à la première transaction de se finir.

## 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.

- 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`).
- 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.

**Réponse:** Voir le projet Eclipse.

## 4 Utilisation à distance des bases du PUIO

On suppose que vous êtes sur votre machine personnelle, sur laquelle vous avez installé Eclipse et Java, et que le TP est correctement importé.

Dans votre code, modifiez le nom d'hôte du serveur de bases de données (généralement dans le `main` ou dans le constructeur de la classe) pour le définir comme `localhost` à la place de `tp_postgres`.

### 4.1 Sous Linux ou MacOS/X

Ouvrir un terminal et rentrer la commande suivante :

```
$ ssh -L 5432:tp-postgres:5432 prenom.nom@ssh1.pgip.universite-paris-saclay.fr
```

en remplaçant `prenom.nom` par votre login au PUIO. Ceci crée un tunnel TCP over SSH entre le port 5432 de votre machine et le port 5432 de la machine `tp-postgres` du PUIO.

### 4.2 Sous Windows

Il faut dans un premier temps récupérer le logiciel `putty.exe` à l'adresse :

<https://www.putty.org/>

Puis cliquer sur le premier lien `Download Putty`. Sur la page de téléchargement, récupérer le fichier 64-bit x86: `putty.exe` dans la section `Alternative binary files`. Vous pouvez le déposer où vous souhaitez et cliquer dessus pour le lancer (c'est un seul exécutable sans besoin d'installation).

Une fois le programme lancé, dans les préférences (barre de gauche) déplier :

- Connection
- SSH
- Tunnels

et remplir les champs suivants :

**Source port :** 5432

**Destination :** `tp-postgres:5432`

Revenir dans la section « `Session` » (la première des préférences) et rentrer dans « *Host Name (or IP address)* » : `ssh1.pgip.universite-paris-saclay.fr` puis cliquer `Open`. Utiliser son login et mot de passe du PUIO.