

# Systèmes de gestion de bases de données

## Anomalies transactionnelles

P. Rigaux

Cnam, dépt. informatique

November 18, 2016

## Mise à jour perdues

C'est **le** cas d'anomalie permis par **tous** les systèmes, en mode transactionnel par défaut.

Retenir: deux transactions **lisent** une même données, et **l'écrivent** ensuite: une des écritures est perdue.

Exemple d'exécution concurrente avec deux transactions de réservation

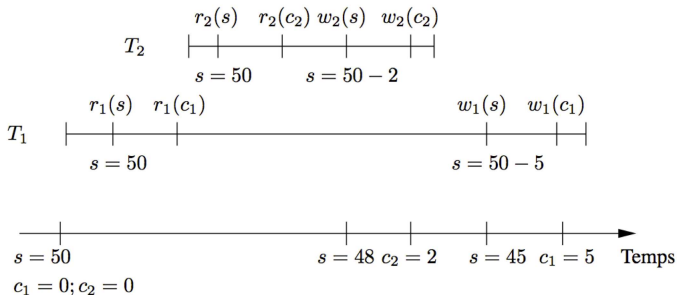
$$r_1(s)r_1(c_1)r_2(s)r_2(c_2)w_2(s)w_2(c_2)w_1(s)w_1(c_1)$$

On effectue d'abord les lectures pour  $T_1$ , puis les lectures pour  $T_2$  enfin les écritures pour  $T_2$  et  $T_1$  dans cet ordre.

# L'anomalie

Point de départ:

- il reste 50 places libres pour le spectacle  $s$ ,  $c_1$  et  $c_2$  n'ont pour l'instant réservé aucune place;
- $T_1$  veut réserver 5 places pour  $s$ ;
- $T_2$  veut réserver 2 places pour  $s$ .



## Le déroulement

Pas à pas, voici ce qui se passe.

- 1  $T_1$  lit  $s$  et  $c_1$ . Nb places libres : 50.
- 2  $T_2$  lit  $s$  et  $c_2$ . Nb places libres : 50.
- 3  $T_2$  écrit  $s$  avec nb places =  $50 - 2 = 48$ .
- 4  $T_2$  écrit le nouveau compte de  $c_2$ .
- 5  $T_1$  écrit  $s$  avec nb places =  $50 - 5 = 45$ .
- 6  $T_1$  écrit le nouveau compte de  $c_1$ .

À l'arrivée, 5 places réservées, 7 places payées. **Incohérence.**

## Pour bien comprendre

À la fin de l'exécution, il reste 45 places vides sur les 50 initiales alors que 7 places ont effectivement été réservées et payées.

**Cette anomalie ne survient qu'en cas d'entrelacement défavorable.** Dans la plupart des cas, une exécution concurrente ne pose pas de problème.

**Le programme est correct.** On peut le regarder 1000 fois, le tester 10000 fois sans jamais détecter d'anomalie.

### Essentiel

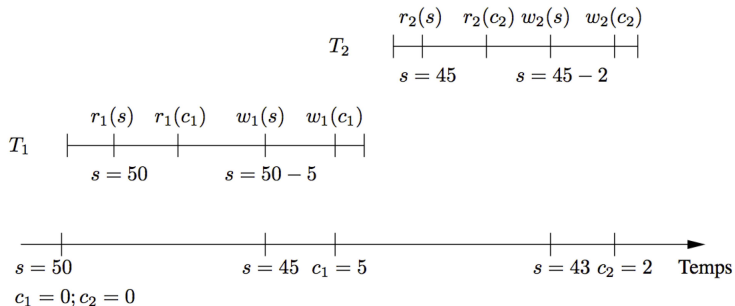
Si on ne sait pas qu'une anomalie de concurrence est possible, l'erreur est incompréhensible.

## Solution radicale: exécution en série

Si on force l'exécution en série:

$$r_1(s)r_1(c)w_1(s)w_1(c)r_2(s)r_2(c)w_2(s)w_2(c)$$

On est sûr qu'il n'y a pas de problème:

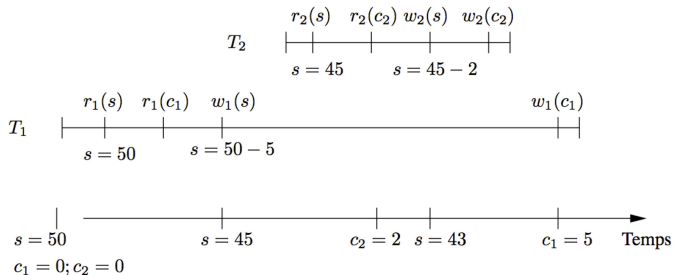


**Très pénalisant..** Une petite transaction (je suis sur mon écran de saisie, je vais déjeuner) bloque tout le monde.

## Les exécutions concurrentes sont possibles

Un exemple qui ne pose pas de problème:

$$r_1(s)r_1(c_1)w_1(s)r_2(s)r_2(c_2)w_2(s)w_1(c_1)w_2(c_2)$$



**Exécution dite sérialisable** car résultat équivalent à l'exécution en série.

### Le contrôle de concurrence

Le contrôle de concurrence doit assurer des exécutions concurrentes sérialisables.

## Lectures non répétables et autres fantômes

Deuxième catégorie d'anomalies. Prenons l'exemple du programme Contrôle.

```
Procédure Contrôle()
```

```
Début
```

```
  Lire tous les clients et effectuer la somme des places prises
```

```
  Lire le spectacle
```

```
  SI (Somme(places prises) <> places réservées)
```

```
    Afficher ("Incohérence dans la base")
```

```
Fin
```

Vérifie la cohérence de la base. Une transaction  $T_c$  a la forme:

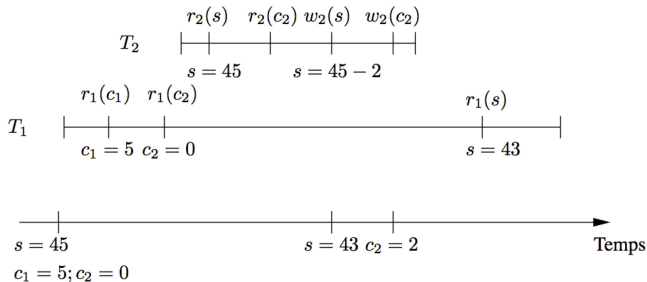
$$r_c(c_1) \dots r_c(c_n) r_c(s)$$



# Une exécution concurrente avec réservation

Entrelacement de  $\text{Contrôle}()$  et  $\text{Res}(c_1, s, 5)$ .

$$r_1(c_1)r_1(c_2)\text{Res}(c_2, s, 2)\dots r_1(c_n)r_1(s)$$



**Le contrôle en déduit (à tort) que la base est incohérente.**

## Que s'est-t-il passé?

Une même transaction (le contrôle) a pu lire deux états **différents** de la base.

On a donc des problèmes liés au manque d'isolation. Deux types:

- La même donnée, lue deux fois de suite, a changé: **lecture non répétable**.
- Des données sont apparues (ou ont disparu): **tuple fantômes**.

Dans les deux cas, isolation partielle (on voit les résultats d'une ou plusieurs autres transactions) et donc risque d'anomalie.

## Un dernier exemple d'anomalie: lectures sales

C'est un autre type de problème (dit "de recouvrabilité"): **l'entrelacement empêche une bonne exécution des commit et rollback.**

Exemple:

$$r_1(s)r_1(c_1)w_1(s)r_2(s)r_2(c_2)w_2(s)w_2(c_2)C_2w_1(c_1)R_1$$

Notez:  $T_2$  a lu la donnée écrite par  $T_1$ ;  $T_2$  valide,  $T_1$  annule.

**Comment gérer cette annulation?**

## Problème insoluble

Annuler  $T_1$ : les mises à jour de  $T_1$  n'ont pas eu lieu.

**Oui mais**,  $T_2$  a lu une des données qui vient d'être effacée.

Alors il faudrait annuler  $T_2$  aussi ? Mais  $T_2$  a fait un `commit`.

Et ainsi de suite... ingérable.

## Bilan des anomalies

Quatre exemples représentatifs, à bien étudier.

- **Mise à jour perdues**: entrelacement rare mais fatal pour la cohérence.
- **Lecture non répétable**: défaut d'isolation.
- **Tuple fantôme**: défaut d'isolation.
- **Lecture sale**: données peut-être cohérente, mais exécution du `commit` et du `rollback` compromise.

**Tout interdire?** Effondrement des performances.

**Les systèmes nous laissent choisir le niveau de protection ...** et autorisent par défaut certaines anomalies.

## Vérifions que nous avons compris

- J'exécute deux fois  
| `select * from Spectacle where id=1`  
dans la même transaction. J'obtiens deux résultats différents. Quelle est la propriété ACID qui n'est pas respectée?
- Je fais la même chose, mais en effectuant un `commit` entre les deux requêtes. J'obtiens deux résultats différents. Quelle est la propriété ACID qui n'est pas respectée?
- Je veux réserver la place 123 sur un vol. Je vérifie qu'elle est libre: oui. Je paye et je réserve. Quand j'arrive dans l'avion, la place est prise par quelqu'un d'autre. Quelle anomalie a pu survenir?
- Un système offre une commande de verrouillage d'une table. Faut-il recommander d'utiliser cette commande pour assurer l'ACIDité des transactions?