Bases de données relationnelles

« *Les transactions* »

G. Raschia

Date de la dernière modification : 16 janvier 2019

Dpt. INFO — Polytech Nantes

Au menu

Reprise sur panne

Intégrité n°1

Correction des données

On veut que les données soient cohérentes ou correctes à tout instant

EMP=

Nom	Âge
Alice	52
Bob	3421
Charlie	1

Intégrité n°2

Cohérence vis-à-vis des contraintes d'intégrité

- Les données doivent satisfaire des prédicats
- Quelques exemples :
  - X est une clé de la relation R
  - La df $X \to Y$ existe dans R
  - $\mathrm{dom}(X) = \{x_1, x_2, x_3\}$
  - α est un index valide sur $R.X$
  - « aucun employé ne doit gagner plus de deux fois le salaire moyen »

Repères

Définitions

- État cohérent : qui satisfait toutes les contraintes
- BdD cohérente : dans un état cohérent
- Résilience : capacité d'une BdD à demeurer cohérente

Propriété requise

A. Cohérence I.D.
Toute transaction respecte les contraintes d'intégrité de la base de données

Une réserve

Correction *vs.* contraintes

La satisfaction de l'ensemble des contraintes déclarées ne signifie pas la « correction intégrale » des données

Exemple (Contrainte de transaction)

- À chaque mise à jour : nouveau salaire $>$ ancien salaire
- À chaque suppression de compte : solde $= 0$

Émulation par des contraintes simples

COMPTE=

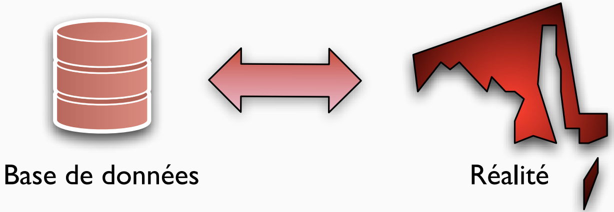
Id	...	Solde	Suppr
1	...	200	F
2	...	-450	F
3	...	0	V
4	...	0	F

- Contrainte : (Suppr = Vrai) $\Rightarrow$ (Solde = 0)

Une autre situation

Exemple (Contraintes implicites)

La base de données est réputée représenter le monde réel...



Résignons-nous

Malgré tout, on continue à utiliser les contraintes d'intégrité

Maintien de la cohérence

$\boxed{\text{Observation}}$ la BdD ne peut pas être cohérente à tout instant !

Exemple

Soit la contrainte $(c_1 + c_2 + \ldots + c_n = \Sigma)$;

- Dépôt de 100€ sur c_2
  1. $c_2 \leftarrow c_2 + 100$
  2. $\Sigma \leftarrow \Sigma + 100$

État transitoire

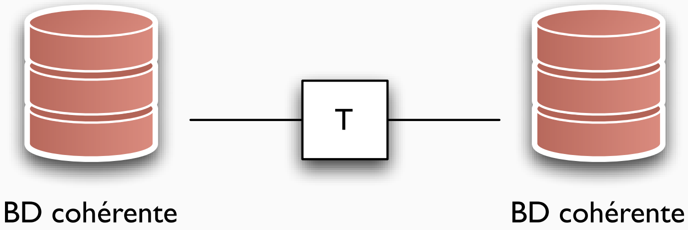
Exemple (suite)

$$
\begin{array}{c|c|c|c|c|c}
\ldots & \ldots & & \ldots & & \ldots \\
c_2 & 50 & \longrightarrow & 150 & \longrightarrow & 150 \\
\ldots & \ldots & & \ldots & & \ldots \\
\Sigma & 700 & & {\color{orange}700} & & 800
\end{array}
$$

Mais alors...

Transaction

Ensemble d'opérations sur la BdD qui préserve sa cohérence



Le b.a.-ba

Hypothèse forte

Si T démarre dans un état cohérent et T s'exécute seule,
alors T termine dans un état cohérent

Un peu plus de sens

Au sujet de la correction

- Si l'on gèle l'exécution des transactions, et après la fin de toute transaction en cours, la BdD est cohérente
- Chaque transaction voit un état cohérent de la BdD

Les problèmes

Quels événements conduisent à violer les contraintes ?

- Erreur de la transaction ou du programme
- Erreur du système de gestion de bases de données
- Panne matérielle
  - crash disque qui altère l'équilibre des comptes
- Partage des données
  - T_1 augmente de 10% le salaire des développeurs
  - T_2 promeut les développeurs en dev-ops

Les solutions

Comment prévenir/corriger les violations de contrainte ?

- Traitement en cas de défaillance (ou panne)
- Traitement en cas de concurrence d'accès
- Traitement conjoint défaillance/concurrence

Hors-champ

Les thèmes qui échappent à notre étude

- Écriture de transactions correctes
- Construction de SGBD corrects
- Validation et réparation de contraintes
  - Les solutions étudiées dans ce chapitre ne requièrent pas la connaissance des contraintes

Les fondements

Le modèle de panne

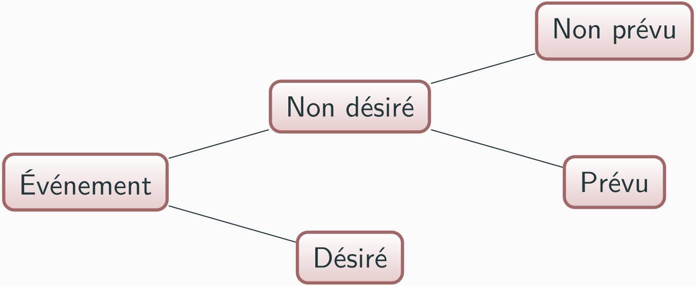
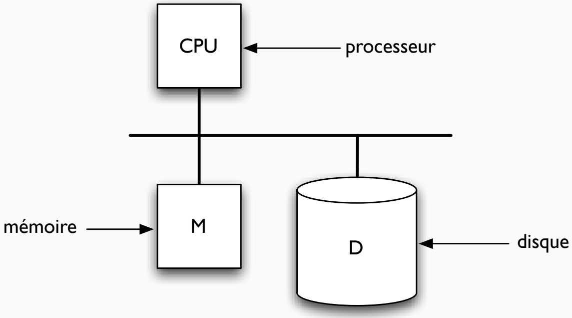


Schéma technique simplifié



Typologie des événements

- Événement désiré : cf. guide utilisateur...
- Événement non désiré mais prévu :
    - Bug de programme : annulation de transaction
    - Erreur système : perte de mémoire, arrêt/réinitialisation CPU

limite de prise en charge

- Événement non désiré et non prévu : Tout le reste !
    - Perte de mémoire sans arrêt CPU
    - Perte de données sur disque
    - Implosion CPU qui anéantit l'univers...

Crédibilité

Ce modèle est-il raisonnable ?

- Approche :
  ajouter des contrôles et de la redondance au système pour augmenter la probabilité de réalisation du modèle
- Par exemple :
  - Réplication des unités de stockage
  - Mémoire avec bit de parité
  - Contrôle du CPU

Second élément déterminant

La hiérarchie des mémoires

De l'un à l'autre

Les opérations

- $\texttt{entrée}(x)$: page contenant $x \rightarrow$ mémoire
- $\texttt{sortie}(x)$: page contenant $x \rightarrow$ disque
- $\texttt{lire}(x, t)$: $\texttt{entrée}(x)$ si nécessaire, puis
  $t \leftarrow$ valeur de x dans la page chargée
- $\texttt{écrire}(x, t)$: $\texttt{entrée}(x)$ si nécessaire, puis
  valeur de x dans la page chargée $\leftarrow t$

Notation : $\texttt{IN}(x)$ $\texttt{OUT}(x)$ $\texttt{R}(x, t)$ $\texttt{W}(x, t)$

Problème clé

Transaction non terminée

Exemple

Soit la contrainte $A = B$;

$$\begin{aligned} T_1 : \quad & A \leftarrow A \times 2 \\ & B \leftarrow B \times 2 \end{aligned}$$

Avec panne

Poursuite de l'exemple

$$
\begin{aligned}
T_1 : \quad & \mathtt{R}(A, t);\ \ t \leftarrow t \times 2 \\
& \mathtt{W}(A, t); \\
& \mathtt{R}(B, t);\ \ t \leftarrow t \times 2 \\
& \mathtt{W}(B, t); \\
& \mathtt{OUT}(A); \\
& \langle\langle \text{ panne ! } \rangle\rangle \\
& \mathtt{OUT}(B);
\end{aligned}
$$

Mémoire

$A : \cancel{8}\ 16$

$B : \cancel{8}\ 16$

Disque

$A : \cancel{8}\ 16$

$B : 8$

Propriété requise

Atomicité C.I.D.

Exécution de toutes les opérations d'une transaction, ou aucune

Une première solution

Journalisation pour « Défaire » (*Undo Logging*)

Principe | Retenir instantanément chaque modification

- Origine : Fil d'Ariane
- Journal des « images Avant »

Edward Burne-Jones, Tile Design - Theseus and the Minotaur in the Labyrinth (1861)

Sur l'exemple

Journalisation instantanée des modifications

Exemple

Soit la contrainte $A = B$;

$$\begin{aligned} T_1 : \quad A &\leftarrow A \times 2 \\ B &\leftarrow B \times 2 \end{aligned}$$

Avec journalisation externe

$T_1:$ $\mathtt{R}(A,t);\ t \leftarrow t \times 2$

$\mathtt{W}(A,t);$

$\mathtt{R}(B,t);\ t \leftarrow t \times 2$

$\mathtt{W}(B,t);$

$\mathtt{OUT}(A);$

$\mathtt{OUT}(B);$

Mémoire

$A : \cancel{8}\ 16$

$B : \cancel{8}\ 16$

Disque

$A : \cancel{8}\ 16$

$B : \cancel{8}\ 16$

Journal

$\langle T_1, \text{start} \rangle$

$\langle T_1, A, 8 \rangle$

$\langle T_1, B, 8 \rangle$

$\langle T_1, \text{commit} \rangle$

Pas si simple

Mécanisme d'écriture

- L'enregistrement du journal est d'abord écrit en mémoire
- Pas d'écriture sur disque à chaque opération

Mémoire

$A : \cancel{8}\ 16$

$B : \cancel{8}\ 16$

$\langle T_1, \text{start} \rangle$

$\langle T_1, A, 8 \rangle$

$\langle T_1, B, 8 \rangle$

Disque

$A : \cancel{8}\ 16$

$B : 8$

Journal

État incorrect #1

Journal incomplet

Solution :

Écriture dans le journal avant la m.à.j. sur disque

Autre problème

Confirmation de la transaction

Mémoire

$A : \cancel{8}\ 16$
$B : \cancel{8}\ 16$
$\langle T_1, \text{start}\rangle$
$\langle T_1, A, 8\rangle$
$\langle T_1, B, 8\rangle$
$\langle T_1, \text{commit}\rangle$

Disque

$A : \cancel{8}\ 16$
$B : 8$

Journal

$\langle T_1, \text{start}\rangle$
$\langle T_1, A, 8\rangle$
$\langle T_1, B, 8\rangle$
$\langle T_1, \text{commit}\rangle$

État incorrect #2

Journal en avance

Solution :

Marque *commit* du journal après les écritures sur disque

Propriété requise

A.C.I. <u>Durabilité</u>

L'effet de toute transaction confirmée (marque `commit`) doit persister

Le protocole

Règles de la journalisation pour « défaire »

1. Pour chaque opération, générer un enregistrement de journal avec l'ancienne valeur de x
2. Avant la m.à.j. de x sur disque, les enregistrements du journal concernant x doivent être écrits sur le disque : principe WAL (*Write-Ahead Logging*)
3. Avant que le *commit* soit ajouté au journal, toutes les écritures de la transaction doivent être réalisées sur le disque

Comment s'en sert-on ?

Règles de reprise avec journalisation pour « défaire »

- Pour chaque T_i ayant un enregistrement $\langle T_i, \text{start}\rangle$ dans le journal :
    - Si $\langle T_i, \text{commit}\rangle$ ou $\langle T_i, \text{abort}\rangle$ alors ne rien faire
    - Sinon

        Pour chaque $\langle T_i, X, v\rangle$ dans le journal : $\texttt{W}(X, v)$; $\texttt{OUT}(X)$

        Écrire $\langle T_i, \text{abort}\rangle$ dans le journal

Est-ce correct ?

Plus exactement

Véritables règles de reprise

1. Soit $S =$ ensemble de transactions avec $\langle T_i, \text{start}\rangle$ dans le journal et ni $\langle T_i, \text{commit}\rangle$ ni $\langle T_i, \text{abort}\rangle$;
   // transactions en cours d'exécution
2. Pour chaque $\langle T_i, X, v\rangle$ du journal, faire dans l'ordre antéchronologique : // plus récent $\rightarrow$plus ancien
   Si $T_i \in S$ alors $\texttt{W}(X, v)$; $\texttt{OUT}(X)$
3. Pour chaque $T_i \in S$, écrire $\langle T_i, \text{abort}\rangle$ dans le journal

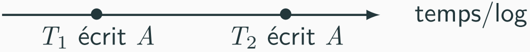
Une dernière précision

Question

Est-ce que les enregistrements $\langle T_i, \text{abort}\rangle$ (étape 3) peuvent être faits dans une ordre quelconque ?

Exemple (T_1 et T_2 écrivent A)

- T_1 est exécuté avant T_2
- T_1 et T_2 subissent un *roll-back*
- $\langle T_1, \text{abort}\rangle$ est écrit mais pas $\langle T_2, \text{abort}\rangle$?
- $\langle T_2, \text{abort}\rangle$ est écrit mais pas $\langle T_1, \text{abort}\rangle$?



Chercher la petite bête

Que se passe-t-il si une panne survient pendant la reprise ?
Aucun problème !

L'opération de reprise à partir d'un journal pour « défaire » est idempotente

$$\mathrm{undo}(\mathrm{undo}) = \mathrm{undo}$$

Le reste à voir

- Journal des « images Après »
- Journal « Avant/Après », quel avantage ?
- Opérations de la vraie vie
- Points de reprise
- Crash disque

Mise-à-jour différée

Journalisation pour « refaire » (*Redo Logging*)

Journal des « images Après »

$$
\begin{aligned}
T_1 : \quad & \texttt{R}(A, t);\ \ t \leftarrow t \times 2 \\
& \texttt{W}(A, t); \\
& \texttt{R}(B, t);\ \ t \leftarrow t \times 2 \\
& \texttt{W}(B, t); \\
& \texttt{OUT}(A); \\
& \texttt{OUT}(B);
\end{aligned}
$$

Mémoire

$A : \cancel{8}\ 16$
$B : \cancel{8}\ 16$

Disque

$A : 8$
$B : 8$

Journal

$\langle T_1, \text{start}\rangle$
$\langle T_1, A, 16\rangle$
$\langle T_1, B, 16\rangle$
$\langle T_1, \text{commit}\rangle$

Journal des « images Après »

$$
\begin{aligned}
T_1 : \quad & \mathtt{R}(A, t);\ \ t \leftarrow t \times 2 \\
& \mathtt{W}(A, t); \\
& \mathtt{R}(B, t);\ \ t \leftarrow t \times 2 \\
& \mathtt{W}(B, t); \\
& \mathtt{OUT}(A); \\
& \mathtt{OUT}(B);
\end{aligned}
$$

Mémoire

$A : \cancel{8}\ 16$
$B : \cancel{8}\ 16$

Disque

$A : \cancel{8}\ 16$
$B : \cancel{8}\ 16$

Journal

$\langle T_1, \text{start} \rangle$
$\langle T_1, A, 16 \rangle$
$\langle T_1, B, 16 \rangle$
$\langle T_1, \text{commit} \rangle$

Journal des « images Après »

$T_1:$ $\mathtt{R}(A, t);\ t \leftarrow t \times 2$

$\mathtt{W}(A, t);$

$\mathtt{R}(B, t);\ t \leftarrow t \times 2$

$\mathtt{W}(B, t);$

$\mathtt{OUT}(A);$

$\mathtt{OUT}(B);$

Mémoire

$A : \cancel{8}\ 16$
$B : \cancel{8}\ 16$

Disque

$A : \cancel{8}\ 16$
$B : \cancel{8}\ 16$

Journal

$\langle T_1, \text{start}\rangle$
$\langle T_1, A, 16\rangle$
$\langle T_1, B, 16\rangle$
$\langle T_1, \text{commit}\rangle$
$\langle T_1, \text{end}\rangle$

Le mécanisme

Règles de la journalisation pour « refaire »

1. Pour chaque opération, générer un enregistrement de journal avec la nouvelle valeur de x
2. Avant la m.à.j. de x sur disque, les enregistrements du journal concernant une transaction ayant modifié x, dont les *commit*, doivent être écrits sur le disque
3. Enregistrer le journal sur disque lorsqu'une transaction est commise : principe FLaC (*Force Logging at Commit*)
4. Écrire un enregistrement $\langle T_i, \text{end} \rangle$ après que les modifications de la BD ont été réalisées sur le disque

Et son usage

Règles de reprise avec journalisation pour « refaire »

- Pour chaque T_i ayant un enregistrement $\langle T_i, \text{commit}\rangle$ dans le journal :
  - Pour chaque $\langle T_i, X, v\rangle$ dans le journal : $\texttt{W}(X, v)$; $\texttt{OUT}(X)$

Est-ce correct ?

Avec un peu de rigueur

Les règles correctes

1. Soit $S =$ ensemble de transactions avec $\langle T_i, \text{commit}\rangle$ (et pas de $\langle T_i, \text{end}\rangle$) dans le journal ;
   // transactions confirmées mais non durables
2. Pour chaque $\langle T_i, X, v\rangle$ du journal, faire dans l'ordre chronologique : // plus ancien $\rightarrow$plus récent
   Si $T_i \in S$ alors `W`(X, v) ; `OUT`(X)
3. Pour chaque $T_i \in S$, écrire $\langle T_i, \text{end}\rangle$ dans le journal

Cas des objets chauds

Séquence d'enregistrements $\langle T_i, \textbf{end}\rangle$

On souhaite différer l'écriture sur disque pour les objets fréquemment mis à jour

Soit X le solde d'un compte ;	Opérations :	
T_1 : ...m.à.j. X...	$\texttt{W}(X); [\texttt{OUT}(X)]$	// T_1
T_2 : ...m.à.j. X...	$\texttt{W}(X); [\texttt{OUT}(X)]$	// T_2
T_3 : ...m.à.j. X...	$\texttt{W}(X); [\texttt{OUT}(X)]$	// T_3
T_4 : ...m.à.j. X...	$\texttt{W}(X); \texttt{OUT}(X)$	// T_4
	$\langle$end$\rangle$ regroupés	

La solution

Les points de reprise (*checkpoint*)

Au lieu d'écrire dans le journal des enregistrements $\langle T_i, \text{end}\rangle$, faire péridoquement :

1. Ne plus accepter de nouvelles transactions
2. Attendre la fin des transactions en cours d'exécution
3. Écrire tous les enregistrements de journal sur le disque
4. Écrire toutes les modifications sur disque (préserver l'image)
5. Indiquer un point de reprise $\langle \text{CP} \rangle$ dans le journal (sur disque)
6. Reprendre le traitement des transactions

Illustration

Exemple (Que faire lors d'une reprise ?)

Journal des « images Après » (sur disque)

...
$\langle T_1, A, 16\rangle$
...
$\langle T_1, \text{commit}\rangle$
...
$\langle \text{CP}\rangle$
...

...
$\langle T_2, B, 17\rangle$
...
$\langle T_2, \text{commit}\rangle$
...
$\langle T_3, C, 21\rangle$
...
« crash ! »

Exemple (Que faire lors d'une reprise ?)

Journal des « images Après » (sur disque)

…
$\langle T_1, A, 16\rangle$
…
$\langle T_1, \text{commit}\rangle$
…
$\langle \text{CP}\rangle$
…

…
$\langle T_2, B, 17\rangle$
…
$\langle T_2, \text{commit}\rangle$
…
$\langle T_3, C, 21\rangle$
…
« crash ! »

$(B \leftarrow 17)$ car T_1 a eu lieu avant $\langle \text{CP}\rangle$ et T_3 n'est pas confirmée

Tout n'est pas rose

Inconvénient du journal des « images Avant »
« *Force* » : besoin d'écrire sur disque dès qu'une transaction est confirmée (marque *commit*)

Inconvénient du journal des « images Après »
« *No Steal* » : besoin de conserver toutes les pages modifiées en mémoire jusqu'à ce que la transaction soit confirmée

Politique d'écriture sur disque

Objectif « *Steal/No Force* »

- Préemption : écriture intermédiaire autorisée
- Libre choix : pas d'écriture systématique à la confirmation

Avantages

1. Performance accrue en fonctionnement normal
2. Aucun contrôle des écritures sur disque

Mélange des genres

Solution : Journal des images « Avant/Après » !
La modification d'un objet X se traduit par un enregistrement de journal de la forme :

$$\langle T_i, X, v\_\text{ancienne}, v\_\text{nouvelle} \rangle$$

Bons principes

Les règles du Undo/Redo

- Un objet X peut être mis à jour sur disque AVANT ou APRÈS la confirmation (*commit*) de la transaction
- **WAL** : les enregistrements de journal sont écrits sur disque AVANT la mise à jour des objets concernés
- **FLaC** : le journal est écrit sur disque à chaque marque *commit*

Et usage

Comment faire la reprise ?

...
$\langle$CP$\rangle$
...
$\langle T_1, A, 9, 10\rangle$
...
$\langle T_1, B, 19, 20\rangle$
...

...
$\langle T_1, \text{commit}\rangle$
...
$\langle T_2, C, 29, 30\rangle$
...
$\langle T_2, D, 39, 40\rangle$
...

« crash ! »

Et usage

Comment faire la reprise ?

...
⟨CP⟩
...
$\langle T_1, A, 9, 10\rangle$
...
$\langle T_1, B, 19, 20\rangle$
...

...
$\langle T_1, \text{commit}\rangle$
...
$\langle T_2, C, 29, 30\rangle$
...
$\langle T_2, D, 39, 40\rangle$
...
« crash ! »

$(D \leftarrow 39, C \leftarrow 29)$ suivi de $(A \leftarrow 10, B \leftarrow 20)$

Point de reprise actif

1. $\langle \text{CP}, \text{start}, (T_1, \ldots, T_k) \rangle$ et écrire le journal sur disque
2. Écrire les pages en suspens (« *dirty pages* ») sur disque
3. $\langle \text{CP}, \text{end} \rangle$ et écrire le journal sur disque

Remarque : $T_1, \ldots, T_k$ sont les transactions actives au point de reprise

Reprise Undo/Redo : phase n°1

Défaire T_1 :

- $B \leftarrow 7$
- $A \leftarrow 9$

...
$\langle T_1, A, 9, 10 \rangle$
...
$\langle \text{CP}, \text{start}, (T_1) \rangle$
...
$\langle \text{CP}, \text{end} \rangle$
...
$\langle T_1, B, 7, 4 \rangle$
...

« crash ! »

Deuxièmement

Reprise Undo/Redo : phase n°2

Refaire T_1 :

- $B \leftarrow 4$
- $C \leftarrow 3$

...
$\langle T_1, A, 9, 10\rangle$
...
$\langle \text{CP}, \text{start}, (T_1)\rangle$
...
$\langle T_1, B, 7, 4\rangle$
...
$\langle \text{CP}, \text{end}\rangle$
...
$\langle T_1, C, 12, 3\rangle$
...
$\langle T_1, \text{commit}\rangle$
...

« crash ! »

Le bon point

D'où reprendre ?

Reprendre au dernier point de reprise valide : CP_1

...
$\langle \mathsf{CP}_1, \mathsf{start}, - \rangle$
...
$\langle \mathsf{CP}_1, \mathsf{end}, - \rangle$
...
$\langle T_1, B, 7, 4 \rangle$
...
$\langle \mathsf{CP}_2, \mathsf{start} \rangle$
...
$\langle T_1, C, 12, 3 \rangle$
...

« crash ! »

Synthèse

Algorithme Undo/Redo de reprise

1. Balayage arrière : fin log $\rightarrow$ début CP valide $+$récent
   - Construire l'ensemble S des transactions confirmées
   - Défaire les actions des autres transactions
2. Marque de début du CP valide :
   - Suivre la chaîne de liens « défaire » pour les transactions dans (liste active)$\backslash S$
3. Balayage avant : début CP valide $+$récent $\rightarrow$ fin log
   - Refaire les actions des transactions de S

Avec des actions dans la vraie vie

Exemple (Retirer de l'argent au DAB)

$$T_i = a_1 a_2 \ldots \mathbf{a_k} \ldots a_n$$

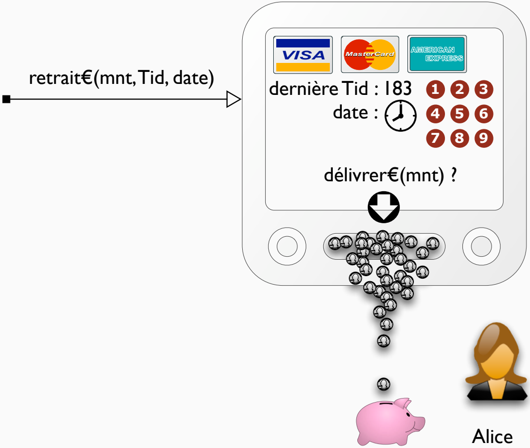
L'action a_k doit délivrer les €€€ !

Bonnes pratiques

Solutions évidentes

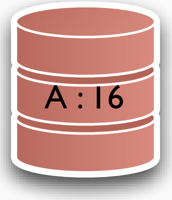
1. Exécuter les actions concrètes APRÈS que la transaction est confirmée
2. Essayer de les rendre idempotentes (en cas de panne/reprise)

Par l'exemple



C'est grave docteur ?

En cas de défaillance du support de stockage stable

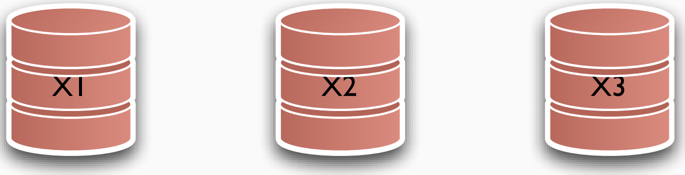


Solution : dupliquer les données !

Voir triple

Exemple n°1 : triple redondance modulaire

- Conserver 3 copies sur disques séparés
- $\texttt{OUT}(X) \rightarrow$ 3 sorties
- $\texttt{IN}(X) \rightarrow$ 3 entrées + vote



Quelque aménagement

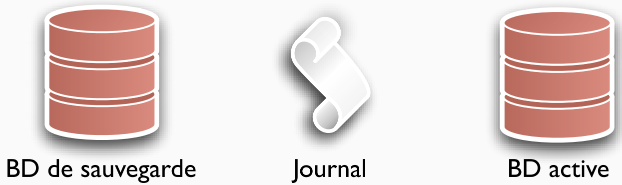
Exemple n°2 : écriture redondante, lecture simple

- Conserver N copies sur disques séparés
- `OUT`$(X) \rightarrow N$ sorties
- `IN`$(X) \rightarrow 1$ entrée
  - Si ok, terminer
  - Sinon, lire une autre copie

Hypothèse : les données erronées sont détectées

Plus probant

Exemple n°3 : *Dump* de la BdD + *Log*

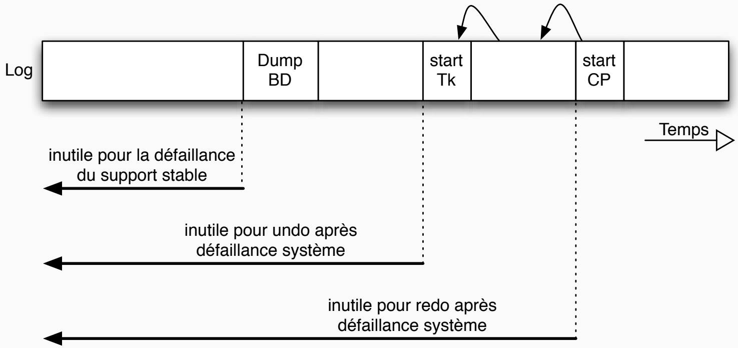


Si la BdD active est perdue :

1. Restaurer la BdD à partir de la sauvegarde
2. Mettre à jour la BdD en rejouant les entrées « refaire » du journal

Les délais

Quand peut-on supprimer les entrées du journal ?



Fin

En résumé

- Cohérence des données
- Une source de problèmes : les défaillances
  - Journalisation
  - Redondance
- Une autre source de problèmes : partage des données

à suivre...

Sérialisabilité

Contrôle de concurrence

- BdD avec contraintes pour la préservation de la cohérence
- T_1, T_2, ..., T_n adressées simultanément à la BdD

Exemple

$$T_1 : \mathtt{R}(A)$$
$$A \leftarrow A + 100$$
$$\mathtt{W}(A)$$
$$\mathtt{R}(B)$$
$$B \leftarrow B + 100$$
$$\mathtt{W}(B)$$

$$T_2 : \mathtt{R}(A)$$
$$A \leftarrow A \times 2$$
$$\mathtt{W}(A)$$
$$\mathtt{R}(B)$$
$$B \leftarrow B \times 2$$
$$\mathtt{W}(B)$$

Contrainte : $A = B$

Plan A

	T_1	T_2
1	$\texttt{R}(A); A \leftarrow A + 100$	
2	$\texttt{W}(A);$	
3	$\texttt{R}(B); B \leftarrow B + 100$	
4	$\texttt{W}(B);$	
5		$\texttt{R}(A); A \leftarrow A \times 2$
6		$\texttt{W}(A);$
7		$\texttt{R}(B); B \leftarrow B \times 2$
8		$\texttt{W}(B);$

A=25	B=25
125	
	125
250	
	250
250	250

Plan B

	T_1	T_2
1		$\mathtt{R}(A); A \leftarrow A \times 2$
2		$\mathtt{W}(A);$
3		$\mathtt{R}(B); B \leftarrow B \times 2$
4		$\mathtt{W}(B);$
5	$\mathtt{R}(A); A \leftarrow A + 100$	
6	$\mathtt{W}(A);$	
7	$\mathtt{R}(B); B \leftarrow B + 100$	
8	$\mathtt{W}(B);$	

A=25	B=25
50	
	50
150	
	150
150	150

Plan C

	T_1	T_2
1	$\mathtt{R}(A); A \leftarrow A + 100$	
2	$\mathtt{W}(A);$	
3		$\mathtt{R}(A); A \leftarrow A \times 2$
4		$\mathtt{W}(A);$
5	$\mathtt{R}(B); B \leftarrow B + 100$	
6	$\mathtt{W}(B);$	
7		$\mathtt{R}(B); B \leftarrow B \times 2$
8		$\mathtt{W}(B);$

A=25	B=25
125	
250	
	125
	250
250	250

Plan E

	T_1	T_2
1	$\texttt{R}(A); A \leftarrow A + 100$	
2	$\texttt{W}(A);$	
3		$\texttt{R}(A); A \leftarrow A \times 2$
4		$\texttt{W}(A);$
5		$\texttt{R}(B); B \leftarrow B \times 2$
6		$\texttt{W}(B);$
7	$\texttt{R}(B); B \leftarrow B + 100$	
8	$\texttt{W}(B);$	

A=25	B=25
125	
250	
	50
	150
250	150

Plan F (= Plan E avec T_{2bis})

	T_1	T_{2bis}	A=25	B=25
1	$\mathtt{R}(A); A \leftarrow A + 100$			
2	$\mathtt{W}(A);$		125	
3		$\mathtt{R}(A); A \leftarrow A \times 1$		
4		$\mathtt{W}(A);$	125	
5		$\mathtt{R}(B); B \leftarrow B \times 1$		
6		$\mathtt{W}(B);$		25
7	$\mathtt{R}(B); B \leftarrow B + 100$			
8	$\mathtt{W}(B);$			125
			125	125

Ce qu'on en retient ?

- Les « bons » plans sont indépendants de
  - l'état initial de la BdD
  - la signification des transactions
- Étude de l'ordre d'apparition des lectures/écritures

Exemple

$$\mathbf{P}_{\mathsf{C}} = r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B)$$

Dans la suite, $\mathtt{R} \equiv r$ et $\mathtt{W} \equiv w$

Plans équivalents

Exemple (De $\mathbf{P}_{\mathsf{C}}$ à $\mathbf{P}_{\mathsf{A}}$)

$$\mathbf{P}_{\mathsf{C}} = r_1(A); w_1(A); \underbrace{r_2(A); w_2(A);}_{\alpha_2} \underbrace{r_1(B); w_1(B);}_{\beta_1} r_2(B); w_2(B)$$

$$\mathbf{P}_{\mathsf{A}} = \underbrace{r_1(A); w_1(A); \underbrace{r_1(B); w_1(B);}_{\beta_1}}_{T_1} \underbrace{\underbrace{r_2(A); w_2(A);}_{\alpha_2} r_2(B); w_2(B)}_{T_2}$$

- Permutation $\alpha_2 \leftrightarrow \beta_1$ légale

Mauvais plan

Un cas épineux : $\mathbf{P}_{\mathsf{E}}$

$$\mathbf{P}_{\mathsf{E}} = \underbrace{r_1(A); w_1(A);}_{\alpha_1} r_2(A); w_2(A); r_2(B); w_2(B); \underbrace{r_1(B); w_1(B)}_{\longleftarrow ?\beta_1}$$

- Vraisemblablement, T_2 doit précéder T_1 car il n'est pas possible de rapprocher β_1 de α_1 dans un plan équivalent
- On note $T_2 \rightarrow T_1$

À propos du Plan E

- $T_2 \to T_1$
- et inversement, $T_1 \to T_2$

En conclusion :

- $\mathbf{P}_\mathsf{E}$ ne peut être réarrangé en $T_1; T_2$ ou $T_2; T_1$
- $\mathbf{P}_\mathsf{E}$ n'est équivalent à aucun plan sériel
- $\mathbf{P}_\mathsf{E}$ est un « mauvais plan »

Retour au Plan C

$$\mathbf{P}_{\mathsf{C}} = \underbrace{r_1(A); w_1(A);}_{\alpha_1} \underbrace{r_2(A); w_2(A);}_{\alpha_2} \underbrace{r_1(B); w_1(B);}_{\beta_1} \underbrace{r_2(B); w_2(B)}_{\beta_2}$$

- $\alpha_1; \alpha_2$ génère une préséance $T_1 \rightarrow T_2$
- $\beta_1; \beta_2$ génère une préséance $T_1 \rightarrow T_2$
- Absence de cycle : $\mathbf{P}_{\mathsf{C}}$ est équivalent à un plan sériel $(T_1; T_2)$

Concepts et définitions 1/2

Transaction

séquence *immuable* d'opérations—actions—$r_i(X)$ et $w_i(X)$

Actions conflictuelles

paire d'actions comprenant au moins une écriture

- $(r_i(X), w_j(X))$ lecture unique *unrepeatable read*
- $(w_i(X), r_j(X))$ lecture sale *dirty read*
- $(w_i(X), w_j(X))$ mise à jour perdue *lost update*

Concepts et définitions 2/2

Plan (ou histoire)

ordre chronologique dans lequel les actions sont exécutées

Plan sériel (ou plan série)

sans entrelacement des actions de transactions distinctes

Propriété requise

A.C. <u>Isolation</u> D.

Toute transaction s'exécute indépendamment des effets de transactions simultanées

Les transactions concurrentes

	T_1	T_2
1	T_1 annonce $\mathtt{R}(A, t)$	
2		T_2 annonce $\mathtt{W}(B, s)$
3		BdD annonce $\mathtt{IN}(B)$
4	BdD annonce $\mathtt{IN}(A)$	
5		$\mathtt{IN}(B)$ terminé
6	$\mathtt{IN}(A)$ terminé	
7		$B \leftarrow s$
8		BdD annonce $\mathtt{OUT}(B)$
9	$t \leftarrow A$	
10		$\mathtt{OUT}(B)$ terminé

Du point de vue du gestionnaire de transactions

Effet après analyse

- $\mathbf{P} = \ldots r_1(A) \ldots w_2(B) \ldots$ ou
- $\mathbf{P} = \ldots w_2(B) \ldots r_1(A) \ldots$

Cas moins favorable

Actions concurrentes et conflictuelles

	T_1	T_2
1		début $w_2(A)$
2	début $r_1(A)$	…
3	…	fin $w_2(A)$
4	fin $r_1(A)$	

- On considère que c'est équivalent à $r_1(A); w_2(A)$ ou $w_2(A); r_1(A)$
- Mécanisme de synchronisation des couches basses
- Hypothèse des actions élémentaires (ou atomiques)

Une définition importante

$\mathbf{P}_1$ et $\mathbf{P}_2$ sont des plans équivalents par conflit si $\mathbf{P}_1$ peut être transformé en $\mathbf{P}_2$ par une série de permutations d'actions consécutives non conflictuelles

Une autre définition importante

Un plan est sérialisable par conflit s'il est équivalent par conflit à un plan sériel (quelconque)

Condition nécessaire et suffisante ?

sérialisabilité $\not\Rightarrow$ sérialisabilité par conflit

Exemple (Plan F)

$\alpha_1; \alpha_2; \beta_2; \beta_1$ avec $T_{2\text{bis}} : . \times 1$

- Dans la suite de l'exposé, on ignore la sémantique des transactions
- Seules les actions conflictuelles sont analysées

Graphe de préséance $\mathcal{G}(\mathbf{P})$

- Les nœuds : transactions de $\mathbf{P}$
- Les arcs : $T_i \to T_j$ si
  - $f_i(A)$ et $h_j(A)$ sont des actions dans $\mathbf{P}$
  - $f_i(A) <_{\mathbf{P}} h_j(A)$
  - au moins l'une des deux actions f_i, h_j est une écriture

Lemme

$\mathbf{P}_1$ et $\mathbf{P}_2$ sont équivalents par conflit $\Rightarrow \mathcal{G}(\mathbf{P}_1) = \mathcal{G}(\mathbf{P}_2)$

Preuve

Considérons $\mathcal{G}(\mathbf{P}_1) \neq \mathcal{G}(\mathbf{P}_2)$

$\Rightarrow \exists T_i : T_i \rightarrow T_j$ dans $\mathbf{P}_1$ et non dans $\mathbf{P}_2$

$\Rightarrow \mathbf{P}_1 = \ldots f_i(A) \ldots h_j(A) \ldots$ et

$\mathbf{P}_2 = \ldots h_j(A) \ldots f_i(A) \ldots$, avec f_i et h_j conflictuelles

$\Rightarrow \mathbf{P}_1$ et $\mathbf{P}_2$ ne sont pas équivalents par conflit $\square$

Remarque

$\mathcal{G}(\mathbf{P}_1) = \mathcal{G}(\mathbf{P}_2) \not\Rightarrow \mathbf{P}_1$ et $\mathbf{P}_2$ sont équivalents par conflit

Contre-exemple

$$\mathbf{P}_1 = w_1(A); r_2(A); w_2(B); r_1(B)$$
$$\mathbf{P}_2 = r_2(A); w_1(A); r_1(B); w_2(B)$$

Théorème

$\mathcal{G}(\mathbf{P})$ acyclique $\iff \mathbf{P}$ est sérialisable par conflit

Preuve

($\Leftarrow$) On considère que $\mathbf{P}$ est sérialisable par conflit
$\Rightarrow \exists \mathbf{P}_s : \mathbf{P}_s$ et $\mathbf{P}$ sont équivalents par conflit
$\Rightarrow \mathcal{G}(\mathbf{P}_s) = \mathcal{G}(\mathbf{P})$
$\Rightarrow \mathcal{G}(\mathbf{P})$ est acyclique puisque $\mathcal{G}(\mathbf{P}_s)$ l'est

Suite de la preuve

Rappel : $\mathcal{G}(\mathbf{P})$ acyclique $\Longleftrightarrow$ $\mathbf{P}$ est sérialisable par conflit

($\Rightarrow$) On considère que $\mathcal{G}(\mathbf{P})$ est acyclique

- Transformer $\mathbf{P}$ comme suit :
    1. Identifier un nœud T_1 sans arc incident
    2. Permuter les actions de T_1 pour qu'elles apparaissent en tête $\mathbf{P}_1 = T_1; \langle \text{le reste} \rangle$
    3. Répéter les étapes précédentes jusqu'au plan sériel

2PL

Comment garantir des plans sérialisables ?

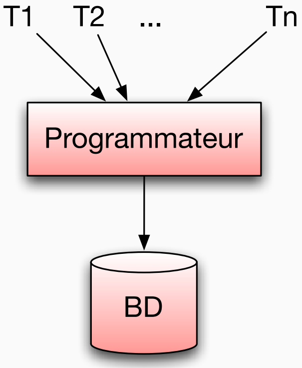
Option n° 1

1. Laisser faire et enregistrer le graphe $\mathcal{G}(P)$
2. Vérifier périodiquement l'absence de cycle
3. Proclamer que l'exécution fut bonne (ou mauvaise)

Un peu plus judicieux

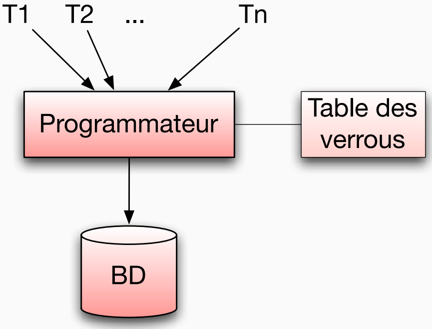
Option nº 2

- Détecter et prévenir les cycles (oui mais comment ?)



Un protocole de verrouillage

- Deux nouvelles actions :
  - verrouiller (verrou exclusif) : $\ell_i(A)$
  - déverrouiller : $u_i(A)$



Règle n$^{\text{o}}$ 1

Transaction bien-formée

$$T_i : \ldots \ell_i(A) \ldots f_i(A) \ldots u_i(A) \ldots$$

Règle n° 2

Plan légal

$$\mathbf{P} = \ldots \ell_i(A) \underbrace{\ldots}_{\neg\ \ell_j(A)} u_i(A)$$

Plan H (Plan E avec verrous)

	T_1	T_2
1	$\ell_1(A); \texttt{R}(A)$	
2	$A \leftarrow A + 100$	
3	$\texttt{W}(A); u_1(A)$	
4		$\ell_2(A); \texttt{R}(A)$
5		$A \leftarrow A \times 2$
6		$\texttt{W}(A); u_2(A)$
7		$\ell_2(B); \texttt{R}(B)$
8		$B \leftarrow B \times 2$
9		$\texttt{W}(B); u_2(B)$
10	$\ell_1(B); \texttt{R}(B)$	
11	$B \leftarrow B + 100$	
12	$\texttt{W}(B); u_1(B)$	

C'est toujours un mauvais plan…

	T_1	T_2	A=25	B=25
1	$\ell_1(A); \mathtt{R}(A)$			
2	$A \leftarrow A + 100$			
3	$\mathtt{W}(A); u_1(A)$		125	
4		$\ell_2(A); \mathtt{R}(A)$		
5		$A \leftarrow A \times 2$		
6		$\mathtt{W}(A); u_2(A)$	250	
7		$\ell_2(B); \mathtt{R}(B)$		
8		$B \leftarrow B \times 2$		
9		$\mathtt{W}(B); u_2(B)$		50
10	$\ell_1(B); \mathtt{R}(B)$			
11	$B \leftarrow B + 100$			
12	$\mathtt{W}(B); u_1(B)$			150
			250	150

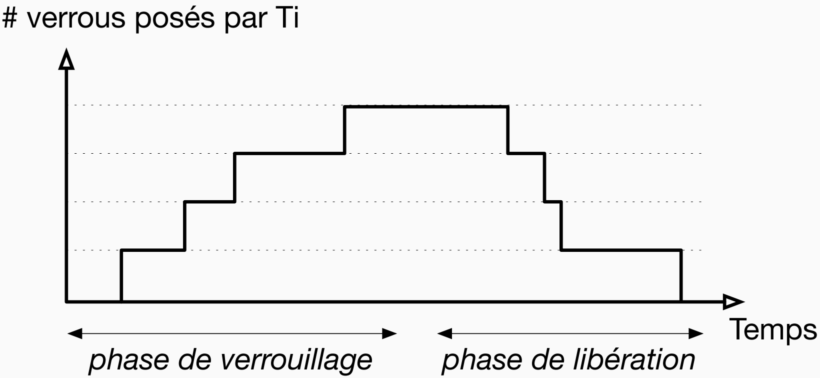
Règle n° 3

Verrouillage à deux phases (2PL)

- Protocole appliqué à une transaction
- (US) *Two Phase Locking*

$$T_i : \quad \underbrace{\ldots \ell_i(A)}_{\text{aucun déverrouillage}} \quad \ldots \quad \underbrace{u_i(A) \ldots}_{\text{aucun verrouillage}}$$

Prise et libération de verrou avec 2PL



Plan K

	T_1	T_2
1	$\ell_1(A); \mathtt{R}(A)$	
2	$A \leftarrow A + 100; \mathtt{W}(A)$	
3	$\ell_1(B); u_1(A)$	
4		$\ell_2(A); \mathtt{R}(A)$
5		$A \leftarrow A \times 2; \mathtt{W}(A)$
		$[\ell_2(B)]$ En attente !

Suite du Plan K

	T_1	T_2
1	$\ell_1(A); \mathtt{R}(A)$	
2	$A \leftarrow A + 100; \mathtt{W}(A)$	
3	$\ell_1(B); u_1(A)$	
4		$\ell_2(A); \mathtt{R}(A)$
5		$A \leftarrow A \times 2; \mathtt{W}(A)$
		$[\ell_2(B)]$ En attente !
6	$\mathtt{R}(B); B \leftarrow B + 100$	
7	$\mathtt{W}(B); u_1(B)$	

Fin du Plan K

	T_1	T_2
1	$\ell_1(A); \mathtt{R}(A)$	
2	$A \leftarrow A + 100; \mathtt{W}(A)$	
3	$\ell_1(B); u_1(A)$	
4		$\ell_2(A); \mathtt{R}(A)$
5		$A \leftarrow A \times 2; \mathtt{W}(A)$
6	$\mathtt{R}(B); B \leftarrow B + 100$	
7	$\mathtt{W}(B); u_1(B)$	
8		$\ell_2(B); u_2(A)$
9		$\mathtt{R}(B); B \leftarrow B \times 2$
10		$\mathtt{W}(B); u_2(B)$

Plan K sérialisable par conflit (equivalent au Plan C !)

Plan L (α_2 et β_2 permutés)

	T_1	$T_{2.1}$
1	$\ell_1(A); \mathtt{R}(A)$	$\ell_2(B); \mathtt{R}(B)$
2	$A \leftarrow A + 100; \mathtt{W}(A)$	$B \leftarrow B \times 2; \mathtt{W}(B)$
3	$[\ell_1(B)]$	$[\ell_2(A)]$

- Pose de verrou mortel (*deadlock*) : « étreinte fatale »
- Situation d'inter-blocage
- $\boxed{\text{Hypothèse}}$ Les transactions en inter-blocage sont annulées (*roll-back*)
  - Leur effet est annulé
  - Elles n'apparaissent plus dans le plan

Exemple

$$\mathbf{P}_{\mathsf{L}} = \underbrace{\qquad\qquad\qquad\qquad}_{\text{rien à signaler !}}$$

Prochaine étape

- Montrer :
  Règles $\#1, 2, 3 \Rightarrow$ Plan sérialisable par conflit

Quelques précisions

- Les situations de conflit pour $\ell_i(A)$ et $u_i(A)$
    - $\ell_i(A)$ et $\ell_j(A)$ sont conflictuelles
    - $\ell_i(A)$ et $u_j(A)$ sont conflictuelles
- Aucun conflit pour
    - $(u_i(A), u_j(A))$
    - $(\ell_i(A), r_j(A))$, ...

Théorème

Règles #1, 2, 3 (2PL) $\Rightarrow$ Plan sérialisable par conflit

Pour les besoins de la preuve

On désigne L(T_i) la première action de la transaction T_i qui libère un verrou

Lemme

$T_i \to T_j$ dans $\mathcal{G}(\mathbf{P}) \Rightarrow \mathsf{L}(T_i) <_{\mathbf{P}} \mathsf{L}(T_j)$

Preuve du lemme

- $T_i \to T_j$ signifie que $\mathbf{P} = \ldots f_i(A) \ldots h_j(A) \ldots$
  avec f et h conflictuelles
- D'après les règles 1 et 2 :
  $\mathbf{P} = \ldots f_i(A) \ldots u_i(A) \ldots \ell_j(A) \ldots h_j(A) \ldots$
- Et d'après la règle 3 : $\mathsf{L}(T_i) \leq_{\mathbf{P}} u_i(A)$ et $\ell_j(A) <_{\mathbf{P}} \mathsf{L}(T_j)$
- Donc $\mathsf{L}(T_i) <_P \mathsf{L}(T_j)$ $\square$

Preuve de théorème

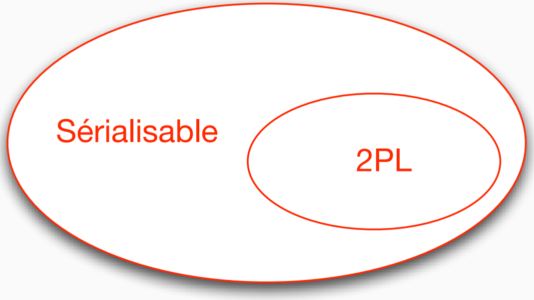
Rappel : Règles #1, 2, 3 (2PL) $\Rightarrow$ Plan sérialisable par conflit

Preuve

1. Considérons que $\mathcal{G}(\mathbf{P})$ possède un cycle $\langle T_1, T_2, \ldots, T_n \rangle$
2. D'après le lemme : $\mathsf{L}(T_1) < \mathsf{L}(T_2) < \ldots < \mathsf{L}(T_1)$
3. Situation impossible, donc $\mathcal{G}(\mathbf{P})$ est acyclique
4. Donc P est sérialisable par conflit $\square$

Plans sur la comète

2PL est une partie (seulement) des plans sérialisables



Sérialisable ET $\neg$ 2PL

Exemple de plan sérialisable et intraitable par 2PL

$$\mathbf{P} = w_1(A); w_3(A); w_2(B); w_1(B)$$

- 2PL n'est pas applicable sur P :
  Le verrou $\ell_1(B)$ survient après $w_2(B)$, donc la libération $u_1(A)$ est également postérieure à $w_2(B)$, ce qui interdit l'exécution de $w_3(A)$
- Néanmoins, $\mathbf{P}$ est sérialisable
  Plan sériel équivalent : $T_2; T_1; T_3$

Verrou partagé

- Jusqu'ici :

$$P = \ldots \ell_1(A); r_1(A); u_1(A) \ldots \ell_2(A); r_2(A); u_2(A) \ldots$$

  $r_1(A)$ et $r_2(A)$ ne sont pas conflictuelles
- Verrouillage alternatif :

$$P = \ldots \ell\text{-s}_1(A); r_1(A); \ell\text{-s}_2(A); r_2(A) \ldots u\text{-}s_1(A); u\text{-}s_2(A)$$

Notations

Actions de verrouillage

- ℓ-$f_i(A)$: verrou sur A en mode $f \in \{s, x\}$
- u-$f_i(A)$: libération de A en mode $f \in \{s, x\}$

Raccourci

- $u_i(A)$: libération de A par T_i, quel que soit le mode de verrouillage

Règle n^{o} 1

$$T_i : \ldots \ell\text{-s}_i(A) \ldots r_i(A) \ldots u_i(A) \ldots$$

$$T_i : \ldots \ell\text{-x}_i(A) \ldots w_i(A) \ldots u_i(A) \ldots$$

Lecture/écriture

Comment traiter le cas de transactions qui lisent et écrivent le même objet ?

Option nº 1

- Verrou exclusif

$$T_i : \ldots \ell\text{-x}_i(A) \ldots r_i(A) \ldots w_i(A) \ldots u_i(A) \ldots$$

Option nº 2

- Promotion : lecture puis peut-être écriture

$$T_i : \ldots \ell\text{-s}_i(A) \ldots r_i(A) \ldots \ell\text{-x}_i(A) \ldots w_i(A) \ldots u_i(A) \ldots$$

- La prise $\ell\text{-x}_i(A)$ est une promotion du verrou $\ell\text{-s}_i(A)$ et peut être assimilée à
  1. soit la prise d'un second verrou ($s \to \{s, x\}$) sur A
  2. soit la séquence $(u_i(A); \ell\text{-x}_i(A))$

Règle n° 2

Plan légal

$$P = \ldots \ell\text{-s}_i(A) \underbrace{\ldots\ldots}_{\neg\,\ell\text{-x}_j(A)} u_i(A) \ldots$$

$$P = \ldots \ell\text{-x}_i(A) \underbrace{\ldots\ldots}_{\neg[\ell\text{-x}_j(A)\vee\ell\text{-s}_j(A)]} u_i(A)) \ldots$$

Résumé de la règle n° 2

Matrice de compatibilité

Comp	S	X
S	Vrai	Faux
X	Faux	Faux

Règle n° 3

Transactions 2PL

- Aucune modification, sauf en cas de promotion
  1. Avec second verrou ($s \rightarrow \{s, x\}$), pas de changement
  2. Avec libération implicite du verrou s ($s \rightarrow x$), l'action est néanmoins autorisée dans la phase de verrouillage

Théorème

Règles #1,2,3 pour les verrous S/X $\Rightarrow$ plan sérialisable par conflit

Preuve

Similaire au cas des verrous X

- Détail :
  - ℓ-$f_i(A); \ell$-$g_j(A)$ ne sont pas conflictuelles si Comp(f, g)
  - ℓ-$f_i(A); w$-$g_j(A)$ ne sont pas conflictuelles si Comp(f, g)

Types de verrous au-delà de s/x

Exemples

1. Verrou d'incrément
2. Verrou de mise-à-jour

Verrou d'incrément

Exemple

- Action d'incrément : in

$$\mathsf{in}_i(A) = \mathtt{R}(A); A \leftarrow A + k; \mathtt{W}(A)$$

- $\mathsf{in}_i(A)$ et $\mathsf{in}_j(A)$ ne sont pas conflictuelles !

$$A = 5 \quad \xrightarrow[+2]{\mathsf{in}_i(A)} \quad A = 7 \quad \xrightarrow[+10]{\mathsf{in}_j(A)} \quad A = 17$$

$$A = 5 \quad \xrightarrow[+10]{\mathsf{in}_j(A)} \quad A = 15 \quad \xrightarrow[+2]{\mathsf{in}_i(A)} \quad A = 17$$

Que devient la matrice de compatibilité ?

Comp	S	X	I
S			
X			
I			

Nouvelle matrice de compatibilité

Comp	S	X	I
S	Vrai	Faux	Faux
X	Faux	Faux	Faux
I	Faux	Faux	Vrai

Verrou de mise-à-jour

Un problème traditionnel de verrou mortel avec les promotions

T_1	T_2
$\ell\text{-s}_1(A)$	
	$\ell\text{-s}_2(A)$
$[\ell\text{-x}_1(A)]$	
	$[\ell\text{-x}_2(A)]$

Verrou mortel !

Une solution

Si T_1 veut lire A et sait par avance qu'elle est susceptible d'écrire A ultérieurement, elle réclame un *verrou de mise-à-jour* (et non un verrou partagé)

- Seuls les verrous de mise-jour peuvent être promus

Et la matrice de compatibilité ?

Comp	S	X	U
S			
X			
U			

- En ligne : les verrous déjà posés
- En colonne (rouge) : les demandes de verrou

Résultat

Comp	S	X	U
S	Vrai	Faux	Vrai
X	Faux	Faux	Faux
U	Faux	Faux	Faux

- Matrice non symétrique

Remarque

- L'objet A peut être verrouillé simultanément dans différents modes

  $$P = \ldots \ell\text{-s}_1(A) \ldots \ell\text{-s}_2(A) \ldots \ell\text{-u}_3(A) \ldots [\ell\text{-s}_4(A)?|\, \ell\text{-u}_4(A)?] \ldots$$

- Pour poser un verrou en mode f, f doit être compatible avec tous les verrous déjà posés sur l'objet

Sujets de discussion

1. Détection
    - temporisation
    - graphe d'attentes
2. Prévention
    - préordonnancement des ressources
    - préordonnancement des transactions

Temporisation

- Si une transaction attend plus de L secondes, on l'annule !

Pros

- Schéma simple

Cons

- Difficile de choisir la valeur de L

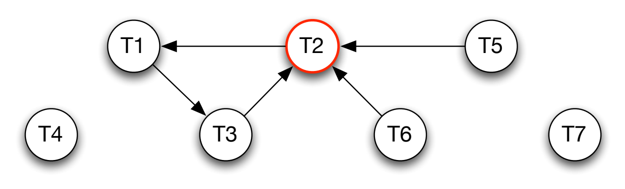
Le graphe d'attentes

Définition constructive

- Les nœud sont les transactions
- Un arc de U à T est formé ssi :
  1. T possède un verrou sur la ressource A ;
  2. U est en attente d'un verrou sur A, et
  3. U ne peut obtenir de verrou sur A dans le mode souhaité qu'à la condition que T ait préalablement libéré A.

Détection de verrous mortels

- Construction du graphe d'attentes
- Utilisation de la table des verrous (posés/en attente)
- Procédé incrémentiel ou périodique
- À la découverte d'un *cycle*, choix et annulation d'une victime...



Prévention des verrous mortels

par préordonnancement des ressources

1. Trier tous les objets A_1, A_2, ..., A_n
2. Une transaction peut verrouiller A_j après A_i seulement si $j > i$

Problème

- La prise ordonnée des verrous n'est pas réaliste et donc peu pratiquée

Prévention des verrous mortels /2

par préordonnancement des transactions

- Schéma *Wait-Die*
- Schéma *Wound-Wait*

Prévention sans réquisition

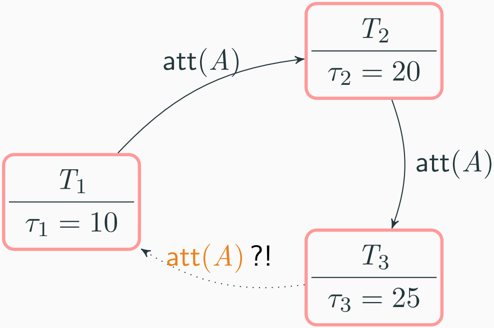
Attendre ou mourir (*wait-die*)

- Les transactions sont estampillées $\tau(T_i)$
- T_j peut attendre que T_i libère une ressource si

$$\tau(T_j) < \tau(T_i)$$

...sinon T_j meurt (elle est annulée)

Exemple



Situation de famine

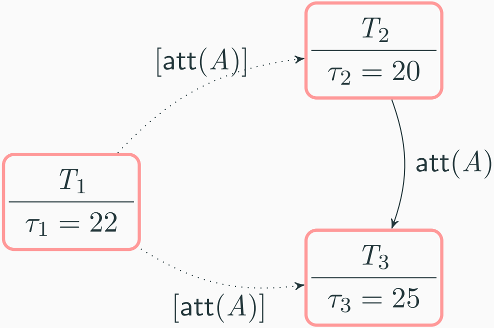
- Lorsqu'une transaction meurt, on la réexécute ultérieurement
- Avec quelle estampille ?
    1. estampille initiale, ou
    2. nouvelle estampille (date de la nouvelle soumission)

Lutte contre la famine

Famine en mode prévention sans réquisition (*wait-die*)

- Nouvelle soumission avec l'*estampille initiale*
- Les garanties contre la famine
  - La transaction d'estampille la plus ancienne ne meurt jamais
  - Une transaction qui meurt (éventuellement plusieurs fois), finit dans le pire cas, par détenir l'estampille la plus ancienne

Deuxième exemple



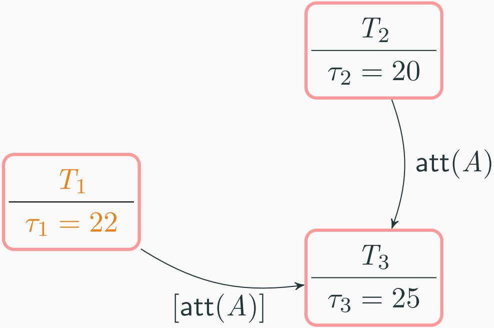
Scénario :

- T_1 réclame A

Remarque :

- $\tau_1 \in [20, 25]$

Option n°1



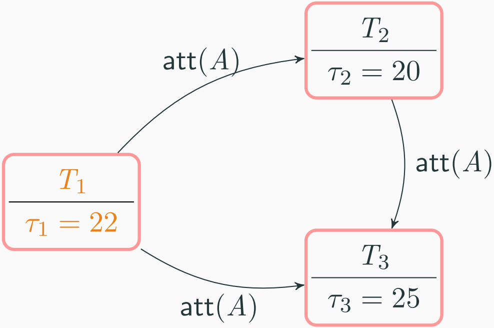
Scénario :

- T_1 réclame A

Déroulement :

1. T_1 réclame A auprès de T_3 qui détient le verrou
2. Lorsque T_2 obtient le verrou, T_1 doit mourir !

Option n°2



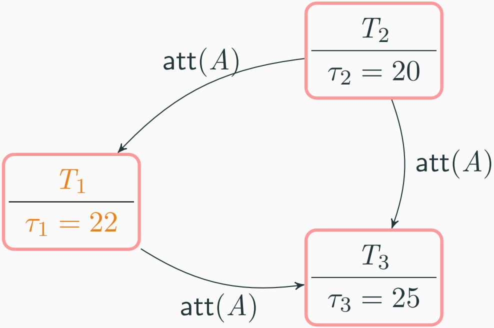
Scénario :

- T_1 réclame A

Déroulement :

1. T_1 obtient le verrou après T_3 puis T_2
2. T_1 meurt sur le champ !

Option n°3



Scénario :

- T_1 réclame A

Déroulement :

1. T_1 *préempte* A au détriment de T_2
2. T_1 réclame A auprès de T_3 seulement
3. T_2 réclame A auprès de T_3 et T_1
4. T_2 devient affamée ?

Prévention avec requisition

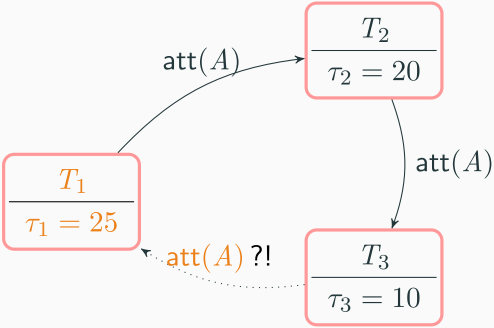
Blesser ou attendre (*wound-wait*)

- Les transactions sont estampillées $\tau(T_i)$
- T_j peut réquisitionner une ressource au détriment de T_i si

$$\tau(T_j) < \tau(T_i)$$

  ...sinon T_j attend
- « blessure » : T_i est annulée et cède le verrou à T_j

Exemple


T_2
$\tau_2 = 20$
$\mathsf{att}(A)$
T_1
$\tau_1 = 25$
$\mathsf{att}(A)$
$\mathsf{att}(A)$?!
T_3
$\tau_3 = 10$

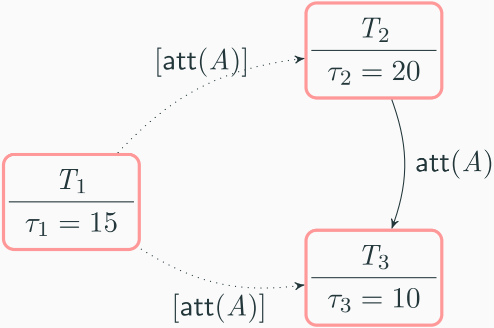
Situation de famine

Quelle estampille après annulation ?

- Avec le schéma *Wound-wait* : on attend les vieilles transactions
- Avec le schéma *Wait-die* : on attend les jeunes transactions
- Quel que soit le schéma, les plus anciennes transactions « tuent » les plus récentes

Conclusion : il faut préserver l'estampille originale pour prévenir la famine !

Deuxième exemple



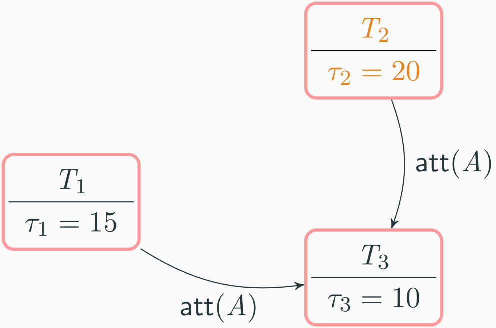
Scénario :

- T_1 réclame A

Remarque :

- $\tau_1 \in [10, 20]$

Option n°1



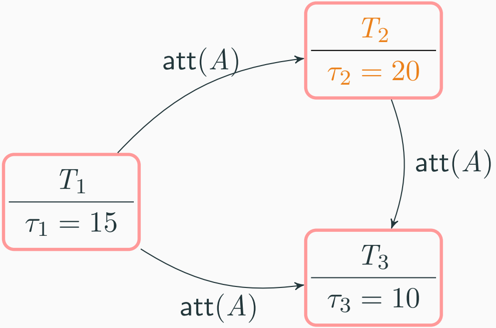
Scénario :

- T_1 réclame A

Déroulement :

1. T_1 réclame A auprès de T_3 qui détient le verrou
2. Lorsque T_2 obtient le verrou, T_1 le réquisitionne et T_2 est « blessée » (donc annulée)

Option n°2



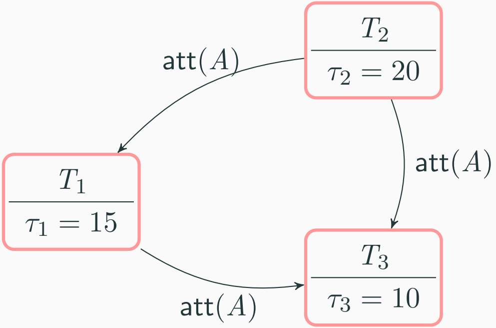
Scénario :

- T_1 réclame A

Déroulement :

1. T_1 obtient le verrou après T_3 puis T_2
2. T_2 est « blessée » immédiatement !

Option n°3



Scénario :

- T_1 réclame A

Déroulement :

1. T_1 préempte A au détriment de T_2
2. T_1 réclame A auprès de T_3 seulement
3. T_2 réclame A auprès de T_3 et T_1
4. T_2 est épargnée !

Verrouillage hiérarchique et arbres

Comment fonctionne le verrouillage en pratique ?

- Chaque système est unique
- Certains systèmes ne garantissent même pas la sérialisabilité par conflit
- Ici : une façon (simplifiée) de considérer le problème

Exemple de système de verrouillage

Les transactions sont ignorantes du mécanisme de verrouillage

1. Prise en charge des demandes et des libérations de verrou
2. 2PL strict : libération des verrous à la confirmation (*commit*) de la transaction

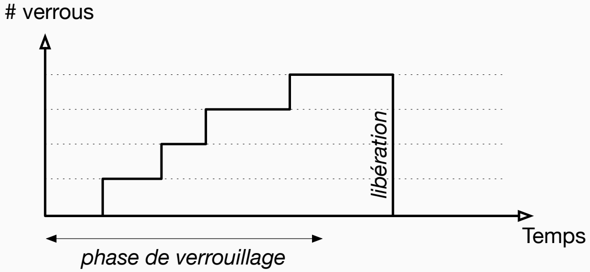
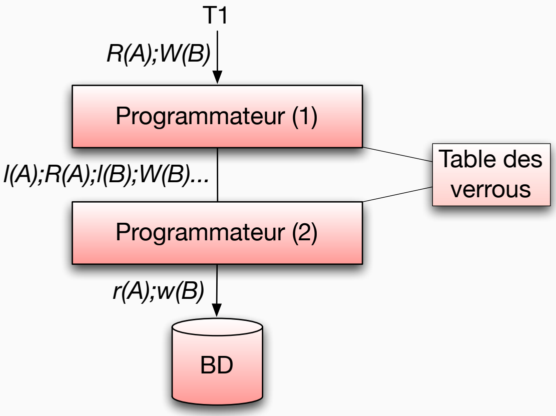
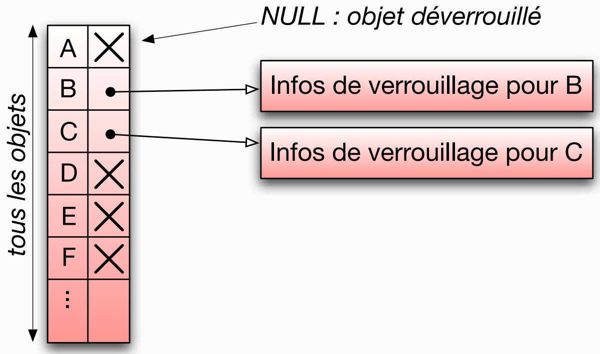


Schéma fonctionnel



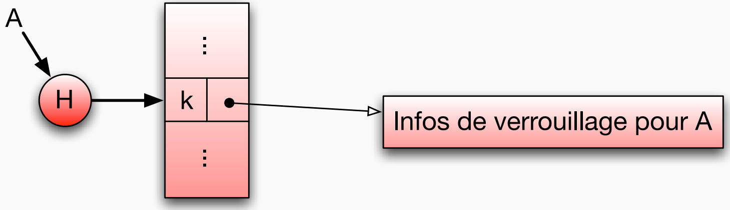
Zoom sur la table des verrous

Point de vue conceptuel



Point de vue logique

Utilisation d'une table de hachage



- Si l'objet est introuvable, alors il est libre

Plus en détail

Exemple : informations de verrouillage pour l'objet A

- Objet : A
- Mode du groupe : U
- En attente : oui
- Liste : cf. liste des transactions

id	trans	mode	att ?	svt	lien
v_1	T_1	S	non	@v_2	@$v_i(T_1)$
v_2	T_2	U	non	@v_3	@$v_j(T_2)$
v_3	T_3	X	oui	$\times$	@$v_k(T_3)$

Nature des objets à verrouiller

BD
Relation A
Relation B
Relation C
⋮

OU

BD
N-uplet A
N-uplet B
N-uplet C
⋮

OU

BD
Page A
Page B
Page C
⋮

?

Taille des objets

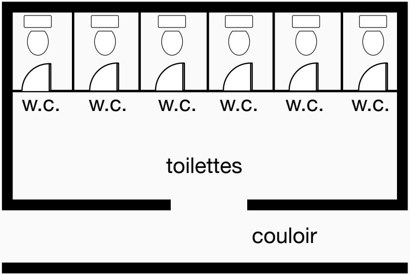
Est-il préférable de choisir des petits ou des gros objets ?

- Le verrouillage fonctionne dans tous les cas
- Choix de gros objets (relations, etc.) :
  - Peu de verrous
  - Peu de concurrence
- Choix de petits objets (n-uplets, attributs, etc.)
  - Plus de verrous
  - Plus de concurrence

Recherche de compromis

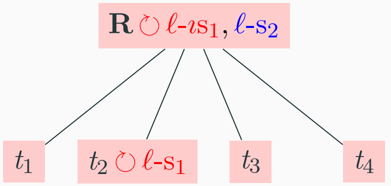
Non ! Il est possible d'avoir les 2

- Toute dame-pipi connait la solution

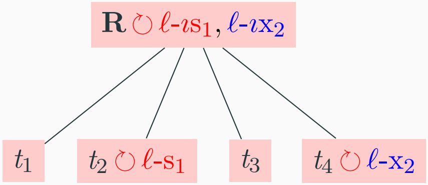


Exemple de verrouillage à granularité multiple

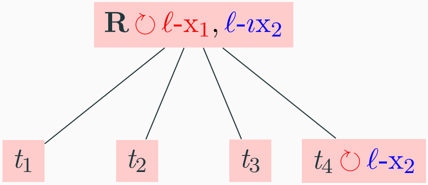
Avec des verrous d'intention



Un autre exemple



Un mauvais exemple



Que devient la matrice de compatibilité ?

Comp	IS	IX	S	SIX	X
IS					
IX					
S					
SIX					
X					

Matrice de compatibilité

Cas du verrouillage à granularité multiple

Comp	IS	IX	S	SIX	X
IS	Vrai	Vrai	Vrai	Vrai	Faux
IX	Vrai	Vrai	Faux	Faux	Faux
S	Vrai	Faux	Vrai	Faux	Faux
SIX	Vrai	Faux	Faux	Faux	Faux
X	Faux	Faux	Faux	Faux	Faux

Une nouvelle dimension

Relation entre le verrou d’un objet composé et le verrou d’un composant

Parent verrouillé en mode	Enfant peut être verrouillé par la même transaction en mode
IS	
IX	
S	
SIX	
X	

Lois de verrouillage Composant/Composé

Parent verrouillé en mode	Enfant peut être verrouillé par la même transaction en mode
IS	IS, S
IX	IS, S, IX, X, SIX
S	[S, IS] inutiles
SIX	X, IX, [SIX]
X	aucun

Règles pour le verrouillage à granularité multiple

1. Respecter le matrice de compatibilité définie pour la granularité multiple
2. Verrou initial sur la racine de l'arbre, quel que soit le mode
3. Le nœud A peut être verrouillé par T_i en mode S ou IS si le parent de A est lui-même verrouillé par T_i en mode IX ou IS
4. Le nœud A peut être verrouillé par T_i en mode X, SIX ou IX si le parent de A est lui-même verrouillé par T_i en mode IX ou SIX
5. T_i adhère au protocole 2PL
6. T_i peut libérer un nœud A à la seule condition que plus aucun fils de A ne soit verrouillé par T_i

Autre problème et sa solution

Opérations d'insertion et de suppression

Exemple :

$$\begin{array}{|c|l}
\cline{1-1}
A & \leftarrow \text{suppression de l'objet A} \\
\vdots & \\
Z & \\
\cline{1-1}
\alpha & \leftarrow \text{insertion de l'objet } \alpha \\
\cline{1-1}
\end{array}$$

Modification des règles de verrouillage

1. Obtenir un verrou exclusif sur A avant sa suppression
2. Lors de l'insertion de α par T_i, T_i obtient systématiquement un verrou exclusif sur α

Encore un problème : les lectures fantômes

Exemple

- Soit une relation $\mathbf{R}$(ID, NOM, …)
- munie de la contrainte d'intégrité : ID est une clé
- et verrouillée par n-uplet

$\mathbf{R}$	ID	NOM	…
t_1	01	de Gaulle	…
t_2	02	Pompidou	…

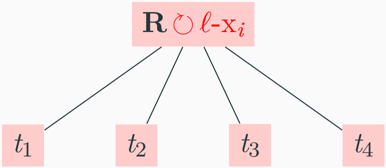
Scénario

- T_1 : insertion de $\langle\ k+1$, VGE, ...$\rangle$ dans $\mathbf{R}$
- T_2 : insertion de $\langle\ k+1$, Mitterrand, ...$\rangle$ dans $\mathbf{R}$

T_1	T_2
$\ell\text{-}\mathrm{s}_1(t_1)$	$\ell\text{-}\mathrm{s}_2(t_1)$
$\ell\text{-}\mathrm{s}_1(t_2)$	$\ell\text{-}\mathrm{s}_2(t_2)$
Vérif. contrainte	Vérif. contrainte
$\vdots$	$\vdots$
Insert. $t_3 = \langle$ 03, VGE, ...$\rangle$	
	Insert. $t_3 = \langle$ 03, Mitterrand, ...$\rangle$

Solution

- Utiliser une hiérarchie d'objets et le verrouillage à granularité multiple
- Avant l'insertion d'un nœud, verrouiller son parent en mode X

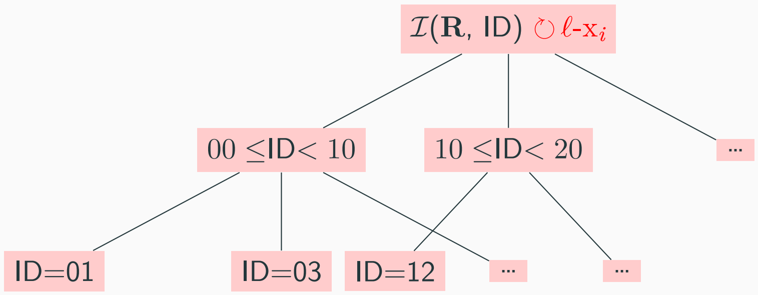


Retour sur l'exemple

	T_1	T_2
01	$\ell\text{-x}_1(\mathbf{R})$	
02		$[\ell\text{-x}_2(\mathbf{R})]$
03	Vérif. contrainte	
04	Insert. $\langle$ 03, VGE, …$\rangle$	
05	$u(\mathbf{R})$	
06		$\ell\text{-x}_2(\mathbf{R})$
07		Vérif. contrainte
08		Oups ! ID=03 déjà dans R !

Verrouillage d'index

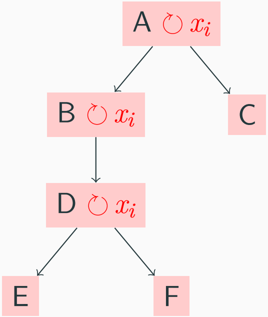
Utiliser l'index plutôt que la relation R



- Verrouillage d'un nœud intermédiaire ?
- Généralisation à plusieurs indexes…

Avec les arbres

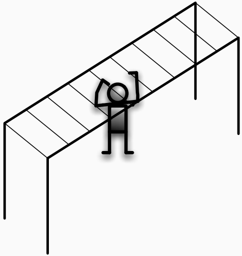
- Dans un *arbre B*, tous les objets sont accessibles à partir de la racine, en suivant les références

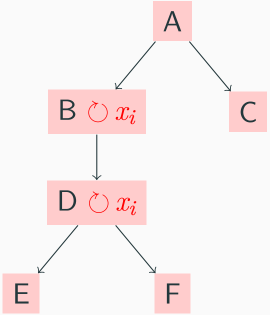


- Peut-on libérer A si l'on n'en a plus besoin ?

Principe

Progression à la façon d'une échelle de traction

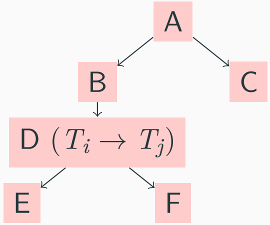




Verrouillage arborescent

Pourquoi ça fonctionne ?

- $\boxed{\text{Hypothèses}}$
  1. Toutes les T_i débutent à la racine
  2. Verrous exclusifs
- $T_i \rightarrow T_j$; donc T_i verrouille la racine avant T_j



- Ça marche même si l'on n'entre pas par la racine

Protocole arborescent avec verrous exclusifs

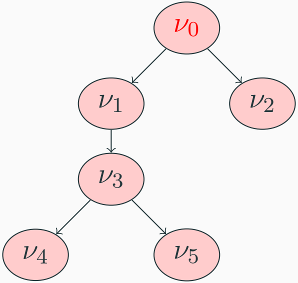
Les règles

1. Le premier verrou de T_i est posé sur un nœud quelconque
2. Ensuite, un nœud A peut être verrouillé si le parent de A est verrouillé par T_i
3. Les nœuds peuvent être libérés n'importe quand
4. Après libération par T_i, un nœud *ne peut plus être verrouillé* par T_i

- Remarques :
  - Violation de 2PL
  - Protocole correct (sûr) car il repose sur le parcours d'arbre

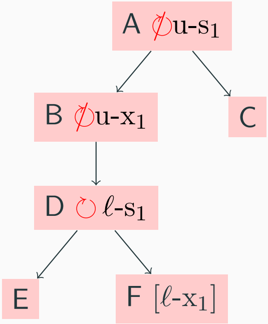
Usage typique du protocole arborescent

Index : contrôle d'accès dans un arbre B/B+

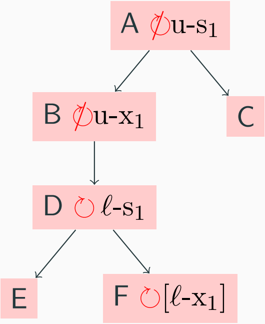


- À l'insertion, le verrou est préservé si le nœud n'est pas sûr
    - pas de dépassement de capacité nécessitant une division

Règles pour verrous partagés et exclusifs ?



Cohabitation des verrous partagés et exclusifs



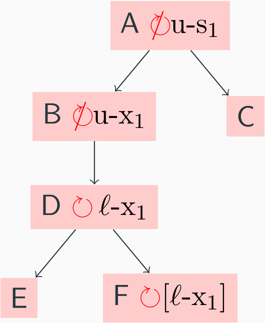
- Lectures de T_2 :
  - Parcours $A \to B \to D \to F$
  - B modifié par T_1
  - F pas encore modifié par T_1

Protocole arborescent avec verrous S et X

Remarques

- Besoin de définir un protocole plus restrictif
  - Une fois que T_1 a verrouillé un objet en mode exclusif, tous les verrous du sous-arbre doivent également être des verrous X
- Et-ce que ça fonctionne ?

Par l'exemple



- Lectures de T_2 :
  - Parcours $A \to B[\to D \to F]$
  - Verrou ℓ-$\mathrm{s}_2(D)$ en attente

Verrouillage physique

À propos de verrouillage

Que verrouille-t-on réellement ?

...

Exemple

Que verrouille-t-on réellement ?

$$T_i : \quad \vdots$$

Lecture du n-uplet t_1

$\vdots$

Mise-à-jour du n-uplet t_2

$\vdots$

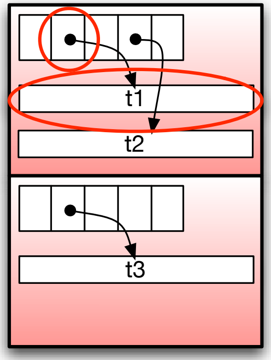
Mise-à-jour du n-uplet t_3

$\vdots$

- Verrouillage au niveau des n-uplets

En pratique

Cuisine interne



Problème :
Le verrou partagé de t_1 risque d'empêcher une mise-à-jour de t_2 (qui pourrait nécessiter une réorganisation de la page)

Solution

Travailler la BdD à deux niveaux

- Niveau supérieur :
  - Actions sur les n-uplets
  - Verrous de n-uplets
  - Actions Refaire/Défaire — logiques
- Exemple pour l'insertion du n-uplet (x, y, z)
  - Refaire : `insert (x,y,z)`
  - Défaire : `delete`

Suite

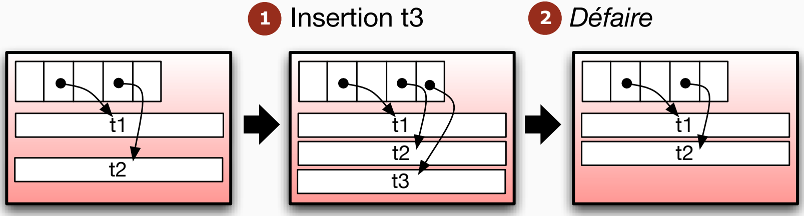
BdD à 2 niveaux

- Niveau inférieur
  - En charge des préoccupations d'organisation physique
  - Verrouillage effectif de la page pendant l'action
  - Et libération après l'action

Identité physique vs. identité logique

Remarque

- Une action « Défaire » ne remet pas la BdD dans son état *physique* originel
- Exemple



Journalisation des actions logiques

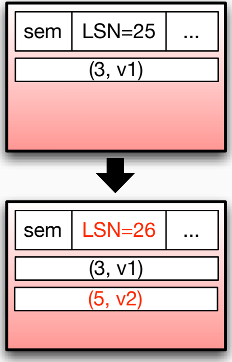
- Une action logique met typiquement en jeu une page entière
- Les entrées du journal des images Avant/Après spécifient des *actions logiques* Défaire/Refaire
- Défi : rendre les actions idempotentes
- Exemple de cas à problème :
  refaire une insertion $\Rightarrow$ clés insérées plusieurs fois !

Solution

Ajout d'un numéro de séquence du journal

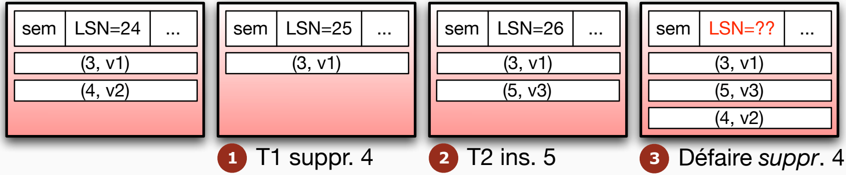
Log Sequence Number

- Enregistrement de journal :
  - LSN $= 26$
  - OP $=$ INSERT $(5, v_2)$ dans page p
  - ...



Oui mais...

Il y a encore un problème !



- Créer une entrée de journal pour l'action défaire : $\text{LSN} = 27$

Enregistrement de compensation

- Enregistrement indiquant les actions « Défaire »
- (inutile pour les actions « Refaire »)
- Remarque : le mécanisme de compensation ne garantit pas l'identité physique de la page

Journal

...
$\langle \mathsf{LSN} = 21, T_1, a_1, p_1 \rangle$
...
$\langle \mathsf{LSN} = 27, T_1, a_2, p_2 \rangle$
...
$\langle \mathsf{LSN} = 35, T_1, a_2^{-1}, p_2 \rangle$
...

Suite de l'exemple

Que faire de p_2 (pendant l'annulation de T_1)

- Si LSN$(p_2) < 27$, alors … ?
- Si $27 \leq$ LSN$(p_2) < 35$, alors … ?
- Si LSN$(p_2) \geq 35$, alors … ?

Remarque : LSN(p_2) est le numéro de séquence du journal affecté à la page p_2 sur disque

Stratégie de reprise avec journalisation logique

$[1]$ **Reconstruire l'état au temps de la panne**

- Trouver le dernier point de reprise valide CKPT, avec $\mathcal{A}$ l'ensemble de ses transactions actives
- Parcourir le journal de CKPT à la fin :
  - Pour chaque entrée de journal $\langle \text{LSN}, p \rangle$, faire
    Si LSN(p) $<$ LSN alors *Refaire*
  - Si l'entrée de journal est début T ou commit T, mettre à jour $\mathcal{A}$

Stratégie de reprise avec journalisation logique (suite)

$[2]$ **Annuler les transactions non confirmées**

- L'ensemble $\mathcal{A}$ contient les transactions à annuler
- Parcourir le journal de la fin à CKPT :
  - Pour chaque entrée de journal (sauf les entrées « défaire ») d'une transaction de $\mathcal{A}$, défaire l'action et générer une entrée « défaire »
  - Pour les transactions de $\mathcal{A}$ non intégralement annulées, lire leurs entrées de journal antérieures à CKPT et défaire les actions (+log)

Exemple : ce qu'il faut faire après une panne

CKPT
...
$\langle \mathsf{LSN} = 21, T_1, a_1, p_1 \rangle$
...
$\langle \mathsf{LSN} = 27, T_1, a_2, p_2 \rangle$
...
$\langle \mathsf{LSN} = 29, T_1, a_3, p_3 \rangle$
...

...
$\langle \mathsf{LSN} = 31, T_1, a_3^{-1}, p_3 \rangle$
...
$\langle \mathsf{LSN} = 35, T_1, a_2^{-1}, p_2 \rangle$
...

Suite de l'exemple

Pendant l'annulation, ne pas considérer les enregistrements pour défaire

- Chaque enregistrement comporte également :
  - Un pointeur arrière vers l'enregistrement précédent de la transaction
- Chaque enregistrement pour défaire comporte également :
  - Un pointeur arrière vers l'enregistrement de l'action correspondante

Idée similaire : les *sagas*

- Activité de longue durée : $T_1; T_2; \ldots T_n$
- À chaque étape/transaction T_i correspond une transaction de compensation T_i^{-1}
- Atomicité sémantique : exécution de l'une des combinaisons suivantes
  - $T_1; T_2; \ldots T_n$
  - $T_1; T_2; \ldots T_{n-1}; T_{n-1}^{-1}; T_{n-2}^{-1}; \ldots T_1^{-1}$
  - $T_1; T_2; \ldots T_{n-2}; T_{n-2}^{-1}; T_{n-3}^{-1}; \ldots T_1^{-1}$
  - ...
  - $T_1; T_1^{-1}$
  - $\emptyset$

Au-delà de la sérialisabilité

Le problème des « données sales »

	T_1	T_2	A=25	B=25
1	$\ell_1(A); r_1(A); A \leftarrow +100$			
2	$w_1(A); \ell_1(B); u_1(A)$		125	
3		$\ell_2(A); r_2(A); A \leftarrow \times 2$		
4		$w_2(A)$	250	
5	$r_1(B)$			
6	ABORT			
7	$u_1(B)$			
8		$\ell_2(B); u_2(A); r_2(B); B \leftarrow \times 2$		
9		$w_2(B); u_2(B)$		50
			250	50

Réparation et contrôle de concurrence

T_i	T_j
$\vdots$	$\vdots$
$w_i(A)$	$\vdots$
$\vdots$	$r_j(A)$
$\vdots$	COMMIT T_j
ABORT T_i	$\vdots$

- COMMIT non persistant
- T_j n'est pas durable !

Solution : les plans réparables

Un autre problème

T_i	T_j
$\vdots$	$\vdots$
$w_i(A)$	$\vdots$
$\vdots$	$r_j(A)$
$\vdots$	$w_j(B)$
ABORT T_i	$\vdots$
$\vdots$	[COMMIT T_j]

- Annulation en chaîne !

Solution : les plans de prévention des annulations en chaîne

Et pourtant...

Plan sérialisable par conflit

- $T_i \rightarrow T_j$

Néanmoins, ce n'est pas un plan réparable

Les besoins

- Prendre une décision « définitive » pour chaque transaction :
  - Confirmation (*commit*) : le système certifie que la transaction s'est terminée ou va se terminer, quelles que soient ses actions
  - Annulation (*abort*) : le système certifie que la transaction a été ou sera abandonnée, son effet sur la BD étant annulé

Actions de plan

Introduction de deux actions supplémentaires

- c_i : la transaction T_i est confirmée
- a_i : la transaction T_i est annulée

Retour sur l'exemple

T_i	T_j
$\vdots$	$\vdots$
$w_i(A)$	$\vdots$
$\vdots$	$r_j(A)$
$\vdots$	$\vdots$
$\vdots$	c_j

Peut-on confirmer au point c_j ?

Lecture et dépendance

Définition préliminaire

T_j lit T_i dans P, noté $T_i \Rightarrow_P T_j$ si

1. $w_i(A) <_P r_j(A)$
2. $a_i \not<_P r_j(A)$
3. si $[w_i(A) <_P w_k(A) <_P r_j(A)]$, alors $(a_k <_P r_j(A))$

Définition importante

Plan réparable (*recoverable*)

Un plan P est réparable à la condition suivante :

Si $(T_i \Rightarrow_P T_j) \ \wedge \ (i \neq j) \ \wedge \ (c_j \in P)$ alors $(c_i <_P c_j)$

Remarque

Les lectures/écritures précèdent la confirmation/annulation

- Si $c_i \in T_i$, alors
  - $r_i < c_i$
  - $w_i < c_i$
- Si $a_i \in T_i$, alors
  - $r_i < a_i$
  - $w_i < a_i$
- Une seule action de type a ou c par transaction

Bonne question

Comment élaborer des plans réparables ?

Avec 2PL

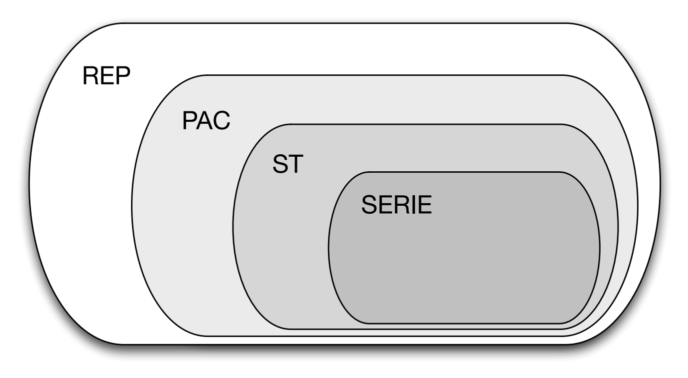
- Retenir les verrous exclusifs jusqu'à la fin de la transaction
- Protocole 2PL strict

T_i	T_j
$\vdots$	$\vdots$
$w_i(A)$	$\vdots$
$\vdots$	$\vdots$
c_i	$\vdots$
$u_i(A)$	$\vdots$
$\vdots$	$r_j(A)$

Un point sur les plans

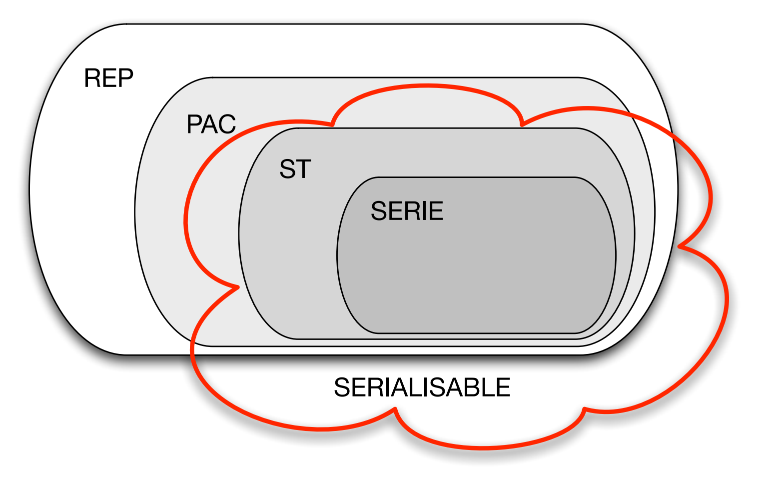
- Plan réparable : toute transaction est *confirmée* après la confirmation des transactions qu'elle lit
- Plan de prévention des annulations en chaîne : toute transaction ne *lit* que des objets écrits par des transactions confirmées
- Plan strict : toute transaction ne *lit* ou n'*écrit* que des objets écrits par des transactions confirmées

Schéma de plans



- Mais où sont les plans sérialisables ?

Schéma de plans /2



Pour fixer les idées

Exemples (avec écritures aveugles)

- plan REP :
  - $w_1(A); w_1(B); w_2(A); r_2(B); c_1; c_2$
- plan PAC :
  - $w_1(A); w_1(B); w_2(A); c_1; r_2(B); c_2$
- plan ST :
  - $w_1(A); w_1(B); c_1; w_2(A); r_2(B); c_2$

Protocoles optimistes

Pessimiste vs. optimiste

Les protocoles pessimistes (verrouillage)

- empêchent les histoires non sérialisables
- ne provoquent pas d'annulation pour cause de non sérialisabilité, mais peuvent en provoquer en cas d'étreinte fatale
- conviennent particulièrement aux charges de travail avec beaucoup de concurrence

Les protocoles optimistes

- supposent l'histoire sérialisable
- provoquent des annulations sur détection de conflits
- conviennent particulièrement aux charges de travail comportant peu de concurrence

Les approches optimistes

1. Validation — *Validation*
2. Horodatage unique — *Timestamp Ordering*
3. Horodatage multiple — *Multi-Version Timestamp Ordering*
4. Isolation d'Instantané — *Snapshot Isolation*

Structure des transactions

Chaque transaction est divisée en 3 étapes :

1. Lecture
   - Toutes les valeurs de la BdD sont lues
   - Écriture sur support temporaire
   - Pas de verrouillage
2. Validation
   - Sérialisabilité du plan ?
   - Si non, annulation des transactions (*rollback*)
3. Écriture
   - Si la validation est positive, écriture dans la BdD

Invariant pour la validation

Rendre *atomique* l'étape de validation

Si $(T_1$, T_2, T_3, ...$)$ est la séquence de validation, alors le plan sera *équivalent par vue* à $P_S = T_1; T_2; T_3; \ldots$

Assure la *sérialisabilité par vue*

En pratique

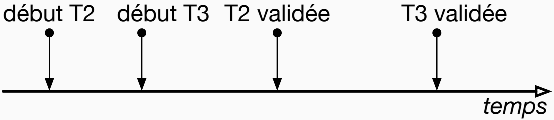
Implémentation du protocole de validation

- Le système doit maintenir trois ensembles
  - START : transactions démarrées, i.e. n'ayant pas atteint la phase 2
  - VAL : transactions confirmées, i.e. ayant passé la phase 2
  - FIN : transactions terminées, i.e. ayant passé la phase 3

Exemple de ce que la validation doit prévenir

$$\mathcal{R}(T_2) = \{B\} \qquad \mathcal{R}(T_3) = \{A, B\}$$

$$\mathcal{W}(T_2) = \{B, D\} \qquad \mathcal{W}(T_3) = \{C\}$$



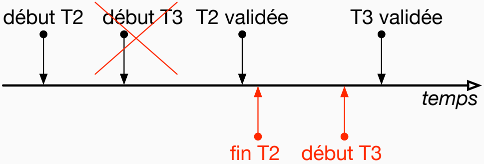
Situation défavorable :

$$\mathcal{W}(T_2) \cap \mathcal{R}(T_3) \neq \emptyset$$

Exemple de ce que la validation doit autoriser

$$\mathcal{R}(T_2) = \{B\} \qquad \mathcal{R}(T_3) = \{A, B\}$$

$$\mathcal{W}(T_2) = \{B, D\} \qquad \mathcal{W}(T_3) = \{C\}$$



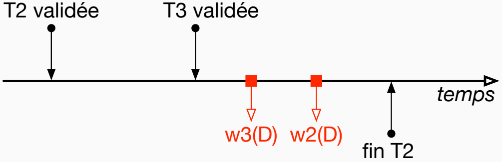
Situation défavorable :

$$\mathcal{W}(T_2) \cap \mathcal{R}(T_3) \neq \emptyset$$

Autre situation que la validation doit prévenir

$$\mathcal{R}(T_2) = \{A\} \qquad \mathcal{R}(T_3) = \{A, B\}$$

$$\mathcal{W}(T_2) = \{D, E\} \qquad \mathcal{W}(T_3) = \{C, D\}$$

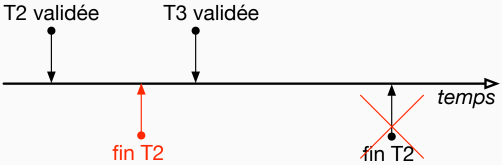


Situation défavorable :

$$\mathcal{W}(T_2) \cap \mathcal{W}(T_3) \neq \emptyset$$

Autre situation que la validation doit autoriser

$$\mathcal{R}(T_2) = \{A\} \qquad \mathcal{R}(T_3) = \{A, B\}$$
$$\mathcal{W}(T_2) = \{D, E\} \qquad \mathcal{W}(T_3) = \{C, D\}$$



Situation défavorable :

$$\mathcal{W}(T_2) \cap \mathcal{W}(T_3) \neq \emptyset$$

Règles de validation pour T_j

1. Lorsque T_j démarre la phase n°1 : IGNORE(T_j) $\leftarrow$ FIN
2. À la validation de T_j :
   Si validée(T_j) alors
   VAL $\leftarrow$ VAL $\cup\{T_j\}$;
   opérer la phase n°3 ; // écriture
   FIN $\leftarrow$ FIN $\cup\{T_j\}$

Fonction de validation

Validée(T_j) **opération atomique !**

Pour $T_i \in$ VAL$\backslash$IGNORE(T_j), faire

Si $(\mathcal{W}(T_i) \cap \mathcal{R}(T_j) \neq \emptyset)$ ou $(T_i \notin$ FIN$)$

Alors RETURN Faux ;

RETURN Vrai ;

- Cette validation est-elle trop restrictive ?

Validation améliorée

Validée(T_j)

Pour $T_i \in$ VAL$\setminus$IGNORE(T_j), faire

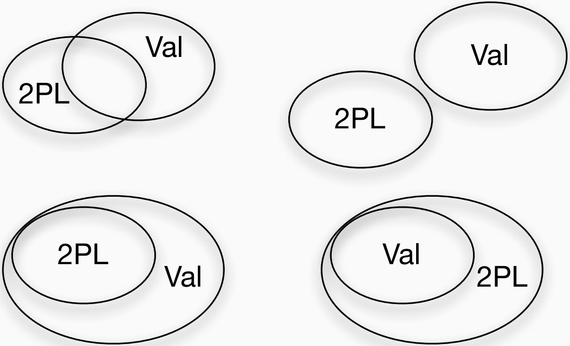
Si $(\mathcal{W}(T_i) \cap \mathcal{R}(T_j) \neq \emptyset)$ ou

(($T_i \notin$ FIN) et $(\mathcal{W}(T_i) \cap \mathcal{W}(T_j) \neq \emptyset)$)

Alors RETURN Faux ;

RETURN Vrai ;

Validation = 2PL ?



On commence par un contre-exemple

$P = w_2(B); w_1(A); w_2(A)$

- P est 2PL :

  $\ell_2(B); w_2(B); \ell_1(A); w_1(A); u_1(A); \ell_2(A); w_2(A); u_2(B); u_2(A)$

- P est invalide :
  Le point de validation de T_2 (val_2) doit être positionné avant $w_2(B)$. Et étant donné le conflit sur A, $\mathsf{val}_1 < \mathsf{val}_2$. Cela conduit à $P = \mathsf{val}_1; \mathsf{val}_2; w_2(B); w_1(A); w_2(A)$
  Pour respecter le protocole de validation, les écritures de T_2 ne devraient pas débuter avant que toutes les écritures de T_1 aient été réalisées, ce qui n'est manifestement pas le cas

Validation est un sous-ensemble de 2PL ?

Idée de preuve (à vérifier !)

- Soit P un plan par validation
- Ajout des verrous pour chaque transaction (P devient P')
    - début T : demande de verrou S pour $\mathcal{R}(T)$
    - validation T : demande de verrou X pour $\mathcal{W}(T)$ et libération des verrous S pour les objets en lecture seule
    - fin T : libération des verrous X
- Les transactions sont alors bien-formées et 2PL
- Il faut encore prouver que P' est légal…

P' est-il légal ?

- Raisonnement par l'absurde : soit P' illégal ;
  1. $P' = \ldots \ell_1(A) \ldots w_2(A) \ldots r_1(A) \ldots \mathsf{val}_1 \ldots u_2(A) \ldots$
     - Au point val_1 : $T_2 \notin$ IGNORE(T_1) ; $T_2 \in$ VAL
     - T_1 est invalide car $\mathcal{W}(T_2) \cap \mathcal{R}(T_1) \neq \emptyset$
     - contradiction !
  2. $P' = \ldots \mathsf{val}_1 \ldots \ell_1(A) \ldots w_2(A) \ldots w_1(A) \ldots u_2(A) \ldots$
     - On considère que T_2 valide en premier (preuve similaire dans les autres cas)
     - Au point val_1 : $T_2 \notin$ IGNORE(T_1) ; $T_2 \in$ VAL
     - T_1 est invalide car ($T_2 \notin$ FIN) et ($\mathcal{W}(T_1) \cap \mathcal{W}(T_2) \neq \emptyset$)
     - contradiction !

Horodatage unique

Timestamp Ordering (TO)

- Chaque transaction reçoit une estampille $\tau(T)$
- L'estampille est fournie par :
  - l'horloge du système, ou
  - un compteur unique, incrémenté par le programmateur

Invariant pour l'horodatage

Sérialisabilité
L'ordre des estampilles, i.e. la chronologie, détermine la séquence des transactions dans le plan série équivalent

Conséquence
Le plan produit est *équivalent par vue* à un plan série. Il est également réparable (et même PAC).

Les marques du protocole

À chaque objet X sont associés :

- $RT(X) =$ l'estampille de la dernière transaction qui a lu X
- $WT(X) =$ l'estampille de la dernière transaction qui a écrit X
- $C(X) =$ l'indicateur de confirmation (*commit bit*) : à `vrai` si la dernière transaction qui a écrit X est confirmée

Principe directeur

Pour chaque action $r_T(X)$ ou $w_T(X)$, vérifier au préalable les conflits :

- $w_U(X) \ldots r_T(X)$: comment déterminer si la lecture est périmée ?
- $r_U(X) \ldots w_T(X)$: l'écriture est-elle périmée ?
- $w_U(X) \ldots w_T(X)$

Idée force

Lorsque T réclame $f_T(X)$, il faut s'assurer que $\tau(U) \leq \tau(T)$

Lecture périmée

T veut lire X

$$\text{start}(T) \ldots \text{start}(U) \ldots w_U(X) \ldots r_T(X)$$

Règle n°1

Si $WT(X) > \tau(T)$, alors T doit être annulée !

Écriture périmée

T veut écrire X

$$\text{start}(T) \ldots \text{start}(U) \ldots r_U(X) \ldots w_T(X)$$

Règle n°2

Si $RT(X) > \tau(T)$, alors T doit être annulée !

La loi de Thomas

Thomas' Write Rule

L'écriture est valable dans le cas suivant :

T veut écrire X

$$\text{start}(T) \ldots \text{start}(U) \ldots w_U(X) \ldots w_T(X)$$

Règle n°3 : loi d'écriture de Thomas
Si $RT(X) \leq \tau(T)$ et $WT(X) > \tau(T)$, alors il est urgent de ne rien faire !

La *sérialisabilité par vue* est assurée avec la loi de Thomas

Le protocole par horodatage

Une transaction T réclame une lecture de X

Si $WT(X) > \tau(T)$, alors annuler T

Sinon, lire X et mettre-à-jour $RT(X) = \max(\tau(T), RT(X))$

Une transaction T réclame une écriture de X

Si $RT(X) > \tau(T)$, alors annuler T

Sinon si $WT(X) > \tau(T)$, alors ignorer l'écriture et poursuivre (loi de Thomas)

Sinon, écrire X et mettre-à-jour $WT(X) = \tau(T)$

Si une transaction vient à être annulée—autrement que par le protocole lui-même—, le protocole échoue à garantir la *réparabilité*

Produire des plans REP/PAC

Rappel

Un plan PAC impose qu'une lecture ne soit acceptée que lorsque l'écriture précédente provient d'une transaction confirmée

Ingrédient supplémentaire

L'indicateur de confirmation $C(X)$ permet exactement de conserver la trace de la confirmation—ou non—de la dernière transaction qui a écrit X

La garantie PAC 1/2

Lecture sale

- T veut lire X et $WT(X) < \tau(T)$
- Les conditions semblent réunies, cependant...

$$\text{start}(U)\ldots\text{start}(T)\ldots w_U(X)\ldots\mathbf{r_T(X)}\ldots\text{abort}(U)$$

Si $C(X) = \texttt{Faux}$, alors T doit attendre qu'il passe à `Vrai`

La garantie PAC 2/2

Révision de la loi de Thomas

- T veut écrire X et $WT(X) > \tau(T)$
- Les conditions semblent réunies pour ne rien faire, cependant...

$$\text{start}(T) \ldots \text{start}(U) \ldots w_U(X) \ldots \mathbf{w_T(X)} \ldots \text{abort}(U)$$

Si $C(X) = \texttt{Faux}$, alors T doit attendre qu'il passe à `Vrai`

La plannification par horodatage

Lorsqu'une transaction T réclame $r_T(X)$ ou $w_T(X)$, le programmateur examine $RT(X)$, $WT(X)$, $C(X)$ et décide de réaliser l'une des opérations suivantes :

- accepter la demande, ou
- annuler T—et la rejouer avec une nouvelle estampille—, ou
- faire patienter T jusqu'à ce que $C(X) = \texttt{Vrai}$

L'indicateur de confirmation

4 règles de gestion

À dériver soi-même à partir du fonctionnement présenté

Horodatage avec indicateur de confirmation

Une transaction T réclame une lecture de X

Si $WT(X) > \tau(T)$, alors annuler T

Sinon si $C(X) = \texttt{Faux}$, alors attendre

Sinon, lire X et mettre-à-jour $RT(X) = \max(\tau(T), RT(X))$

Une transaction T réclame une écriture de X

Si $RT(X) > \tau(T)$, alors annuler T

Sinon si $WT(X) > \tau(T)$, alors

 Si $C(X) = \texttt{Faux}$, alors attendre

 Sinon ignorer l'écriture (loi de Thomas)

Sinon, écrire X et mettre-à-jour $WT(X) = \tau(T)$ et $C(X) = \texttt{Faux}$

L’horodatage unique, en résumé

- Sérialisable par vue
- PAC, et *a fortiori* REP
- sensible aux *lectures fantômes*
  - à traiter séparément, par exemple à l’aide de verrous de prédicats

Horodatage multiple

Multi-Version Timestamp Ordering (MVTO)
de la famille *Multi-Version Concurrency Control* (MVCC)

- Lorsqu'une transaction T réclame $r_T(X)$, mais que $WT(X) > \tau(T)$, alors T doit être annulée

Idée force

Conserver des états—*versions*—successifs de X :
X_t, X_{t-1}, X_{t-2}, … avec

$$\tau(X_t) > \tau(X_{t-1}) > \tau(X_{t-2}) > \dots$$

Détails

À l'écriture $w_T(X)$

Créer une nouvelle version, notée X_t avec $t = \tau(T)$

À la lecture $r_T(X)$

Trouver la version X_t la plus récente telle que $t < \tau(T)$

Remarques

- $WT(X_t) = t$ est immuable
- $RT(X_t)$ est conservé, pour vérifier la légalité des écritures

Suppression de X_t ?

Lorsqu'il existe une version plus récente X_{t+k} et que toute transaction active a démarré après $t + k$ i.e., $\tau(T) > t + k$

Exemple (à emporter)

Étant donnés les états X_3, X_9, X_{12} et X_{18}

1. $r_6(X)$: que se passe-t-il ?
2. $w_{14}(X)$: que se passe-t-il ?
3. $r_{15}(X)$: que se passe-t-il ?
4. $w_5(X)$: que se passe-t-il ?

Quand peut-on supprimer X_3 ?

Exemple d'horodatage unique

	T_1	T_2	T_3	T_4	A
$\tau =$	150	200	175	225	$RT = 0,\ WT = 0$
1	$r_1(A)$				$RT = 150$
2	$w_1(A)$				$WT = 150$
3		$r_2(A)$			$RT = 200$
4		$w_2(A)$			$WT = 200$
5			$r_3(A)$		
6			ABORT		
7				$r_4(A)$	$RT = 225$

Exemple d'horodatage multiple

	T_1	T_2	T_3	T_4	A_0	A_{150}	A_{200}
$\tau =$	150	200	175	225	$RT = 0$, $WT = 0$		
1	$r_1(A)$				$RT = 150$		
2	$w_1(A)$					NEW	
3		$r_2(A)$				$RT = 200$	
4		$w_2(A)$					NEW
5			$r_3(A)$			$RT = 200$	
6				$r_4(A)$			$RT = 225$

Isolation d'instantané

Snapshot Isolation (SI)

Un aperçu

- Chaque transaction reçoit une estampille $\tau(T)$
- La transaction T voit une image au temps $\tau(T)$ de la BdD
- Les conflits w/w sont résolus par la » règle de la première confirmation »
  - la transaction perdante est annulée
- Les conflits r/w et w/r sont purement ignorés !
- Lorsque T est confirmée, ses *pages sales* sont écrites sur disque

Détails

- De la famille des MVCC
  - Pour chaque objet X : X_t, X_{t-1}, X_{t-2}, …
- Lorsque T lit X, elle lit en fait $X_{\tau(T)}$
- Lorsque T écrit X, pour prévenir les mises à jour perdues :
  - Si la dernière version de X est $X_\tau(T)$, alors ok
  - Si $C(X) = \texttt{Vrai}$, alors annuler T
  - Si $C(X) = \texttt{Faux}$, alors attendre
- Lorsque T est confirmée, réaliser les écritures sur disque

Sur quelques propriétés

- Pas de lecture sale : pourquoi ?
- Pas de lecture unique : pourquoi ?
- Pas de mise-à-jour perdue, grâce à la règle de la première confirmation

Par ailleurs :

- Aucune lecture n'est reportée

Malgré tout :

- Il subsiste des conflits r/w !

L'écriture biaisée

Write Skew

T_1 :

Lire(X) ; Si $X \geq 50$

Alors $Y = -50$; Écrire(Y)

COMMIT

T_2 :

Lire(Y) ; Si $Y \geq 50$

Alors $X = -50$; Écrire(X)

COMMIT

L'histoire jouée :

$$r_1(X), r_2(Y), w_1(Y), w_2(X), c_1, c_2$$

Avec $X = 50$ et $Y = 50$, l'exécution sous SI donne $X = -50$ et $Y = -50$

Ce plan n'est pas sérialisable !!

Commentaire général

Contrôle de concurrence optimiste

Adapté à certaines situations :

- conflits rares
- abondance de resources allouées au système
- contraintes de temps réel

Compromis

- transactions en lecture seule : TO/MVTO/VAL/SI
- transactions en lecture-écriture : 2PL

Mémo

Toujours se référer à la documentation !

- DB2 : 2PL strict (S2PL)
- SQL Server :
  - S2PL pour les 4 niveaux d'isolation SQL
  - MVCC pour une implémentation de SI
- Oracle : S2PL + SI pour le mode *sérialisable* !
- PostgreSQL : SI et, récemment, *SI sérialisable* (SSI) !

Synthèse générale

Transactions et contrôle de concurrence

Étude des mécanismes utilisés en pratique

- Journalisation Undo/Redo
- Protocole par verrouillage : 2PL, S2PL
- Granularité multiple
- Protocole arborescent (pour les indexes)
- Protocoles optimistes : TO, MVTO, VAL, SI