

# LA CONCURRENCE

---

Objectifs

Les bases

Le verrouillage deux phases

L'ordonnancement par estampilles

Les applications avancées

©Gardarin 2001

1

## 1. Objectifs

---

- Permettre l'exécution simultanée d'un grand nombre de transactions
- Régler les conflits lecture / écriture
- Garder de très bonne performance
- Eviter les blocages

⟨N°⟩

©Gardarin 2001

2

# Les problèmes de concurrence

## ➤ Perte d'opérations

- { T1 : Read A → a; T2 : Read A → b; T2 : b+1 → b;  
T2 : Write b → A; T1 : a\*2 → a; T1 : Write a → A }

Que contient A ?

## ➤ Introduction d'incohérence

- A = B { T1 : A\*2 → A; T2 : A+1 → A; T2 : B+1 → B; T1 :  
B\*2 → B }

## ➤ Non reproductibilité des lectures

- { T1 : Read A → a; T2 : Read A → b; T2 : b+1 → b; T2 :  
Write b → A; T1 : Read A → a }

⟨N°⟩

©Gardarin 2001

3

## 2. Les bases

### ➤ Chaque transaction $T_i$ est composée d'une séquence d'actions $\langle a_{i1}, a_{i2}, \dots, a_{ini} \rangle$

### ➤ Une exécution simultanée (Histoire) des transactions $\{T_1, T_2, \dots, T_n\}$ est une séquence d'actions

- $H = \langle a_{i1j1}, a_{i2j2}, \dots, a_{ikjk} \rangle$  telle que  $a_{ij} < a_{ij+1}$  pour tout  $i$  et tout  $j$  et quel que soit  $a_{ij}$  de  $T_1, \dots, T_n$ ,  $a_{ij}$  est dans  $H$
- C'est une séquence d'actions complète respectant l'ordre des actions des transactions

### ➤ Une exécution est sérielle si toutes les actions des transactions ne sont pas entrelacées

- elle est donc de la forme
- $\langle Tp(1), Tp(2), \dots, Tp(n) \rangle$  ou  $p$  est une permutation de  $1, 2, \dots, n$ .

⟨N°⟩

©Gardarin 2001

4

# Sérialisabilité

## ➤ Exécution sérialisable

- Une exécution est dite sérialisable si elle est équivalente à une exécution sérielle

## ➤ Plusieurs critères d'équivalence possibles

- Equivalence de vue : tous les résultats visibles sont identiques
- Equivalence du conflit : toutes les actions conflictuelles sont effectuées dans le même ordre sur les objets de la base

⟨N°⟩

©Gardarin 2001

5

# Graphe de précedence

## ➤ Précedences

- Techniques basées sur la seule sémantique des opérations de lecture / écriture
- $T_i$  lit  $O$  avant  $T_j$  écrit  $\Rightarrow T_i$  précède  $T_j$
- $T_i$  écrit  $O$  avant  $T_j$  écrit  $\Rightarrow T_i$  précède  $T_j$
- $T_i$  écrit  $O$  avant  $T_j$  lit  $\Rightarrow T_i$  précède  $T_j$

## ➤ Condition de sérialisabilité

- Le graphe de précedence doit rester sans circuit

⟨N°⟩

©Gardarin 2001

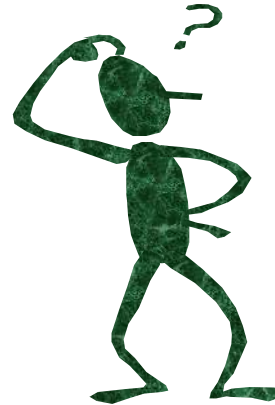
6

## Bilan Problématique

- La sérialisabilité est une condition suffisante de correction

- Exercice

- Démontrer que les cas de perte d'opérations et d'incohérences sont non sérialisables



⟨N°⟩

©Gardarin 2001

7

## 3. Le Verrouillage 2 phases

- PRINCIPES

- verrouillage des objets en lecture/écriture
- opérations Lock(g,M) et Unlock(g)
- compatibilité:

	L	E
L	V	F
E	F	F

- toute transaction attend la fin des transactions incompatibles
- garantie un graphe de précédence sans circuit
- les circuits sont transformés en verrous mortels

⟨N°⟩

©Gardarin 2001

8

# Algorithmes Lock

```
➤ Bool Function Lock(Transaction t, Objet O, Mode M){
  Cverrou := 0 ;
  Pour chaque transaction i ≠ t ayant verrouillé l'objet O faire {
    Cverrou := Cverrou U t.verrou(O) ; // cumuler les verrous sur O
    si Compatible (Mode, Cverrou) alors {
      t.verrou(O) = t.verrou(O) U M; // marquer l'objet verrouillé
      Lock := true ; }
    sinon {
      insérer (t, Mode) dans la queue de O ; // mise en attente de t
      bloquer la transaction t ;
      Lock := false ; } ;
  }
```

⟨N°⟩

©Gardarin 2001

9

# Algorithme Unlock

```
➤ Procédure Unlock(Transaction t, Objet O){
  t.verrou(O) := 0 ;
  Pour chaque transaction i dans la queue de O {
    si Lock(i, O,M) alors {
      enlever (i,M) de la queue de O ;
      débloquer i ; } ;
  }
```

⟨N°⟩

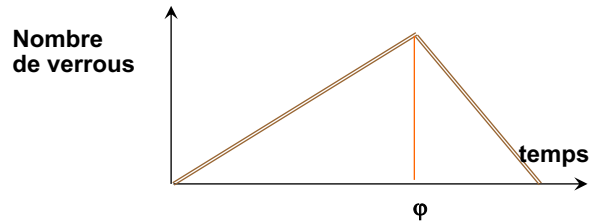
©Gardarin 2001

10

# Condition de corrections

## ➤ Transactions deux phases

- une transaction ne peut relâcher de verrous avant de les avoir tous acquis



⟨N°⟩

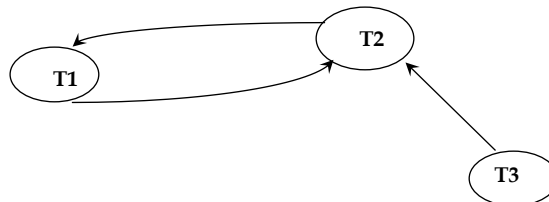
©Gardarin 2001

11

# Problèmes du Verrouillage

## ➤ Verrou mortel

- risques de circuit d'attentes entre transactions



## ➤ Granularité des verrous

- page : en cas de petits objets, trop d'objets verrouillés
- objet : trop de verrous, gestion difficile

⟨N°⟩

©Gardarin 2001

12

# Résolution du verrou mortel

## ➤ Prévention

- définir des critères de priorité de sorte à ce que le problème ne se pose pas
- exemple : priorité aux transactions les plus anciennes

## ➤ Détection

- gérer le graphe des attentes
- lancer un algorithme de détection de circuits dès qu'une transaction attend trop longtemps
- choisir une victime qui brise le circuit

©Gardarin 2001

⟨N°⟩

13

# Améliorations du verrouillage

## ➤ Relâchement des verrous en lecture après opération

- - non garantie de la reproductibilité des lectures
- + verrous conservés moins longtemps

## ➤ Accès à la version précédente lors d'une lecture bloquante

- - nécessité de conserver une version (journaux)
- + une lecture n'est jamais bloquante

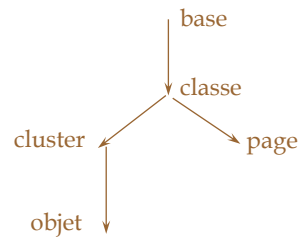
©Gardarin 2001

⟨N°⟩

14

# Granularité Variable

- Plusieurs granules de verrouillage sont définis, inclus l'un dans l'autre
- Le verrouillage s'effectue en mode intention sur les granules supérieurs et en mode effectif sur les granules choisis
  - les modes intentions sont compatibles
  - les modes effectifs et intentions obéissent aux compatibilités classiques



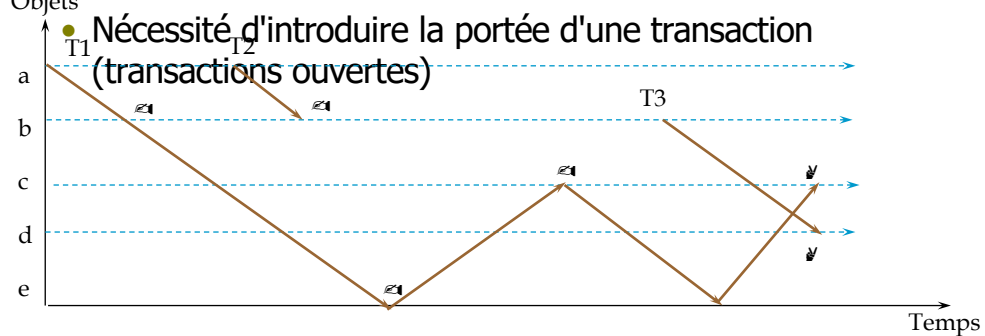
⟨N°⟩

©Gardarin 2001

15

# Verrouillage Altruiste

- Restitution des verrous sur les données qui ne seront plus utilisées
  - L'abandon d'une transaction provoque des cascades d'abandons



⟨N°⟩

©Gardarin 2001

16



# Degré d'isolation en SQL2

## ➤ Définition de degrés d'isolation emboîtés

- Degré 0
  - garantit la non perte des mises à jour
  - pose de verrous courts exclusifs lors des écritures
- Degré 1
  - garantit la cohérence des mises à jour
  - pose de verrous longs exclusifs en écriture
- Degré 2
  - garantit la cohérence des lectures individuelles
  - pose de verrous courts partagés en lecture
- Degré 3
  - garantit la reproductibilité des lectures
  - pose de verrous longs partagés en lecture

⟨N°⟩

©Gardarin 2001

17

# Bilan Verrouillage

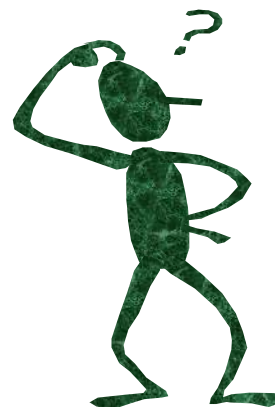
## ➤ Approche pessimiste

- prévient les conflits
- assez coûteuse
- assez complexe

## ➤ Approche retenue

- dans tous les SGBD industriels

## ➤ Difficile de faire mieux !



⟨N°⟩

©Gardarin 2001

18

## 4. Ordonnancement par estampillage

### ➤ Estampille (TimeStamp) associée à chaque transaction

- date de lancement de la transaction
- garantie d'ordre total (unicité)

### ➤ Conservation des estampilles

- dernier écrivain : **Writer**
- plus jeune lecteur : **Reader**

### ➤ Contrôle d'ordonnancement

- en écriture: estampille écrivain  $>$  Writer et  $>$  Reader
- en lecture: estampille lecteur  $>$  Writer

### ➤ Problèmes

- reprise de transaction en cas d'accès non sérialisé
- risque d'effet domino en cas de reprise de transaction

⟨N°⟩

©Gardarin 2001

19

## Algorithme d'ordonnancement

### ➤ Fonction READ (Transaction t, Objet O) {

```
    si  $O.Writer \leq t.estampille$  alors {  
        Read := Get(O) ; // effectuer la lecture  
        O.Reader := MAX (O.Reader, t.estampille) ; // mettre à  
                                                    jour dernier lecteur }  
    sinon Abort(t); }
```

### ➤ Fonction Write (Transaction t, Objet O, Contenu C){

```
    si  $O.Writer \leq t.estampille$  et  $O.Reader \leq t.estampille$  alors {  
        Set(O,C) ; // effectuer l'écriture  
        O.Writer := t.estampille ; // mettre à jour dernier  
                                                    écrivain }  
    sinon Abort(t); }
```

⟨N°⟩

©Gardarin 2001

20

# La Certification Optimiste

## ➤ Les contrôles s'effectuent seulement en fin de transaction

- Phase d'accès : on garde les OID des objets lus/écrits
- Phase de certification : on vérifie l'absence de conflits (L/E ou E/E même objet) avec les transactions certifiées pendant la phase d'accès
- Phase d'écriture (commit) pour les transactions certifiées

## ➤ Avantages et inconvénients

- + test simple d'intersection d'ensembles d'OID en fin de transaction
- - tendance à trop de reprises en cas de conflits fréquents (effondrement)

⟨N°⟩

©Gardarin 2001

21

# Bilan Estampillage

## ➤ Approche optimiste

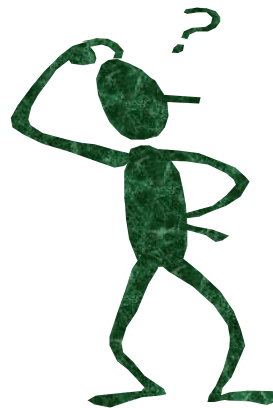
- coût assez faible
- détecte et guérit les problèmes

## ➤ Guérison difficile

- catastrophique en cas de nombreux conflits
- absorbe mal les pointes

## ➤ Sophistication

- ordonnancement multiversions



⟨N°⟩

©Gardarin 2001

22

## 5. Techniques avancées

### ➤ Transactions longues

- mise à jour d'objets complexes
- sessions de conception

### ➤ Prise en compte de la sémantique des application

- opérations commutatives (e.g., ajouts d'informations)
- essais concurrents

### ➤ Travail coopératif

- modèles concurrents plutôt que séquentiels

⟨N°⟩

©Gardarin 2001

23

## Commutativité d'Opérations

### ➤ Possibilité de distinguer d'autres opérations que Lire/Ecrire

- chaque objet possède sa liste d'opérations (méthodes)
- les opérations commutatives n'entraînent pas de conflits
- la commutativité peut dépendre du résultat

### ➤ Cas des ensembles

	[Ins,ok]	[Del,ok]	[In, true]	[In, False]
[Ins,ok]	1	0	0	0
[Del,ok]	0	1	0	1
[In, true]	0	0	1	1
[In, False]	0	1	1	1

⟨N°⟩

©Gardarin 2001

24

# Contrôleur de Commutativité

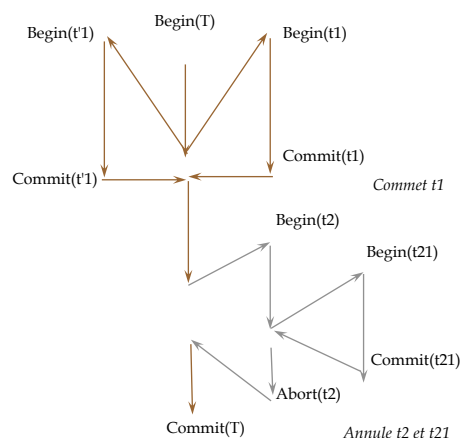
- Chaque objet possède un contrôle de concurrence défini au niveau de la classe
  - laisse passer les opérations commutatives
  - bloque les opérations non commutatives (ordonnancement)
- Le modèle est ouvert et nécessite
  - soit des transactions de compensations
  - soit la gestion de listes de transactions dépendantes
- Un potentiel pour les SGBDO non encore implémenté (complexe)

©Gardarin 2001

25

# Transactions Imbriquées

- ## ➤ OBJECTIFS
- Obtenir un mécanisme de reprise multi-niveaux
  - Permettre de reprendre des parties logiques de transactions
  - Faciliter l'exécution parallèle de sous-transactions
- ## ➤ SCHEMA
- Reprises et abandons partiels
  - Possibilité d'ordonner ou non les sous-transactions



©Gardarin 2001

26

## Verrouillage et Transactions Imbriquées

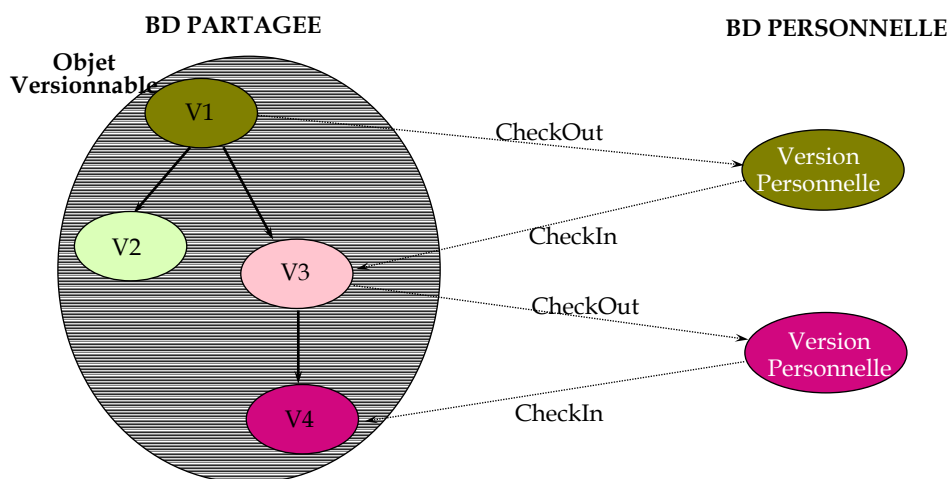
- Chaque transaction peut acquérir ou relâcher des verrous
- Un verrou est accepté si l'objet est libre ou verrouillé par un ancêtre
- Les verrous non relâchés sont rendus à la transaction mère
- Problèmes de conflits entre sous-transactions parallèles
  - risque de verrous mortels
  - l'ancêtre commun peut trancher
- Gestion de journaux multiniveaux
  - organisation sous forme de piles
  - nécessité de journaux multiples en cas de parallélisme

⟨N°⟩

©Gardarin 2001

27

## Versions



⟨N°⟩

©Gardarin 2001

28

## CheckOut / CheckIn

### ➤ CheckOut

- extrait un objet de la BD afin d'en dériver une nouvelle version

### ➤ CheckIn

- réinstalle une nouvelle version de l'objet dans la BD

	Lecture 1	Ecriture 0	COut P 1	COut E 0
Lecture	1	0	1	0
Ecriture	0	0	0	0
COut Partagée	1	0	1	0
COut Exclusif	0	0	0	0

⟨N°⟩

©Gardarin 2001

29

## Fusion des Versions

### ➤ Maintient différentiel

- seuls les objets/pages modifiés sont maintenus

### ➤ Pas d'objets communs modifiés

- la fusion est une union des deltas

### ➤ Des objets communs modifiés

- nécessité d'intervention manuelle (choix)

⟨N°⟩

©Gardarin 2001

30

## 6. Conclusion

---

### ➤ Amélioration du verrouillage

- Transactions ouvertes
- Granularité variable
- Commutativité des opérations
- Transactions imbriquées
- Versions

### ➤ Amélioration des modèles transactionnels

- Transactions imbriquées
- Sagas, Activités, Versions

### ➤ Beaucoup d'idées, peu d'implémentations originales

- la plupart des systèmes utilise le verrouillage type SQL

⟨N°⟩

©Gardarin 2001