La gestion des transactions

La transaction

pour l'utilisateur : ensemble de requêtes (**commandes SQL**) indivisible du point de vue de la cohérence des données, exécuté en “tout ou rien”:

validation (commit) ou annulation (rollback)

pour le SGBD : ensemble d'actions (**lectures/écritures de pages**) à faire puis rendre persistantes (si commit demandé et possible) ou à faire puis défaire (si commit impossible ou rollback demandé) pour exécuter une transaction utilisateur.

Si plusieurs utilisateurs agissent simultanément alors le SGBD doit **ordonnancer** les actions en fonction de celles qui sont en concurrence sur la même page.

Le **rôle du SGBD** = permettre à l'utilisateur :

- d'être sûr que la base reste dans un état cohérent (**intégrité**)
- d'ignorer les actions des autres utilisateurs (**contrôle de concurrence**)
- de ne pas prendre en compte les risques de pannes (**fiabilité**)

=> **Performance :**

nbTransactions/sec = nbUtilisateurs simultanés / Temps cycle d 'une transaction

12 mars 2001 1

Transactions : propriétés ACID

Atomicité : une transaction est soit réalisée entièrement, soit pas du tout.

fin anormale = abandon de transaction ou panne système

Cohérence : une transaction transforme une BD cohérente en une autre BD cohérente.

=> Chapitre précédent : contraintes d’intégrité

Isolation : une transaction ne peut montrer son effet aux autres transactions avant sa validation (-> éviter l'effet domino)

=> ce chapitre 1ère Partie

Durabilité : les effets d'une transaction validée ne seront jamais perdus

(« persistance » // objets)

=> ce chapitre 2e Partie

12 mars 2001 2

Partie 1 : Isolation et Contrôle de concurrence

Accès concurrents à des données
=> peut générer des inconsistances, des pertes d'infos…

Principe de base
Sérialisabilité d'une exécution

Deux techniques
Verrouillage vs. Estampillage

Deux approches
Pessimiste vs. Optimiste

12 mars 2001 3

Sérialisabilité

- action (atomicité)
- permutabilité
- précédence
- validation d'une exécution

T1	LECT(O1)
T2	LECT(O1)
T3	ECR(O2)
T1	LECT(O2)
T2	ECR(O1)

TEMPS

$\not\equiv$

T1	LECT(O1) LECT(O2)
T2	LECT(O1) ECR(O1)
T3	ECR(02)

MAIS

$\equiv$

T3	ECR(02)
T1	LECT(O1) LECT(O2)
T2	LECT(O1) ECR(O1)

Une **exécution** de n transactions T1…Tn est **sérialisable ssi** elle est **équivalente** à une **exécution en série** (ordonnée ou non) des transactions T1…Tn

12 mars 2001 4

Relation de Précédence

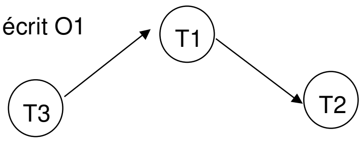
T1 LECT(O1)
T2 LECT(O1)
T3 ECR(O2)
T1 LECT(O2)
T2 ECR(O1)

Ex 1

Dans cet exemple

Sur O1: T1 lit O1, T2 lit O1, T2 écrit O1

Sur O2 : T3 écrit O2, T1 lit O2



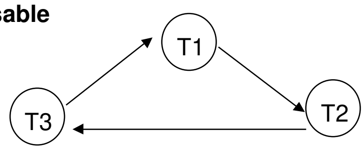
T1 Lect (O1)
T2 Ecrit (O1)
T3 Ecrit (O2)
T3 Ecrit (O1)
T1 Lect (O2)

Ex 2

Dans cet exemple

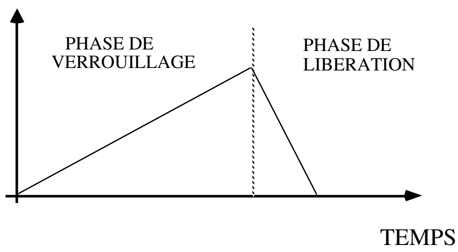
le graphe de précédence a un cycle

l'exécution est non sérialisable



Verrouillage

- **2 modes** de verrouillage :
  exclusif (écriture) ou **partagé** (lecture)
  X **S**
- transactions **bien formées**
  Tous les accès sont **bien déclarés**



- Règles de **compatibilité** des verrous
  en lecture si seulement des verrous en lecture
  en écriture si aucun verrous
- Verrouillage **deux phases** : une transaction ne redemande pas un verrou après en avoir libéré un.
  ->déverrouillage de tous les verrous en écriture après la validation.

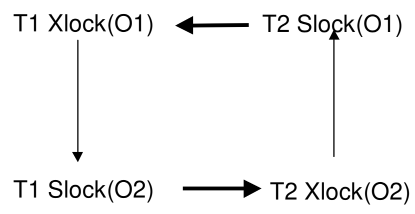
-> Toute exécution complète d'un ensemble de transactions 2-phases est sérialisable

Verrous mortels

Exemple

T1	T2
Xlock(O1)	
	Xlock(O2)
Slock(O2)	
Attente	
	Slock(O1)
	Attente

temps



Entre deux transactions différentes, attente de libération d'une ressource

Au sein de la même transaction : attente de la fin d'une action

12 mars 2001 7

L'estampillage

Principe : l'exécution des transactions doit être équivalente à une exécution en série ordonnée par les estampilles

=> les transactions doivent accéder aux objets dans l'ordre dans lequel elles naissent

- L'estampille d'une transaction est sa date de début
- L'estampille d'un objet est égale à l'estampille de la dernière transaction qui y a accédé.
- Pour accéder à un objet, l'estampille de la transaction doit être supérieure à l'estampille de l'objet

Conséquence

Priorité aux transactions récentes.

Perfectionnement

Distinguer des estampilles en lecture et d'autres en écriture

-> Limiter les conflits

12 mars 2001 8

Algorithme de base

1- Chaque transaction T reçoit une estampille E(T)

2- Chaque opération (R ou W) demandée par la transaction est marquée par l'estampille de la transaction

3- Pour chaque donnée X, 2 estampilles (R et W) qui sont les plus grandes des estampilles des transactions ayant opéré

4- Soit TS l'estampille d'une lecture sur une donnée x

Si TS < W(x) alors la lecture est refusée et la transaction abandonnée

5- Soit TS l'estampille d'une écriture sur une donnée x

Si TS < R(x) ou W(x) alors écriture refusée et transaction abandonnée

12 mars 2001 9

Verrouillage + estampilles : méthodes pessimistes

=> Quand une transaction se met en attente d'une autre transaction, le mécanisme de prévention se met en œuvre. S'il y a un **risque de verrou mortel**, alors une transaction est abandonnée

=> On tue la transaction la plus jeune (Estampillage des transactions)

2 règles possibles. conflit entre Ti et Tj, Ti plus ancienne que Tj (i < j)

- Sans droit de préemption
  - Si Tj attend Ti alors Tj est abandonnée et recommencée avec la même estampille
  - Si Ti attend Tj alors Ti reste en attente
- avec droit de préemption
  - Si Ti attend Tj alors Tj est abandonnée, la ressource est cédée à Ti
  - Si Tj attend Ti alors Tj reste en attente

12 mars 2001 10

Méthodes optimistes

Principe :

On suppose qu'il n'y a pas de conflit pendant l'exécution et on le vérifie en fin de transaction

- Contraire aux autres méthodes (dites pessimistes) qui font les vérifications à priori
- Suppose qu'il y a rarement des conflits. Coût élevé si conflits fréquents

Une transaction = phase de lecture + phase de validation + phase d'écriture

phase de lecture => génère un ensemble d'écritures (write-set)

Si phase de validation -> TRUE alors phase d'écriture sinon restart

12 mars 2001 **11**

Les niveaux d'isolation en SQL (1/2)

Niveau d'isolation	Dirty read	Non repeatable	phanto
Read uncommited	yes	yes	yes
Read committed	no	yes	yes
repeatable read	no	no	yes
serializable	no	no	no

12 mars 2001 **12**

Les niveaux d'isolation en SQL (2/2)

Dirty read :
-> on lit tout, y compris les tuples écrits par des transactions non encore validées.
résultat : on lit des données non valides

Non repeatable read :
-> les lectures sont faites hors transaction.
Si on lit pose deux fois la même requête, on peut avoir deux résultats différents

phantom:
-> associé a un verrouillage tuple
on accède à tous les tuples d'une relation
un tuple est ajouté

12 mars 2001 **13**

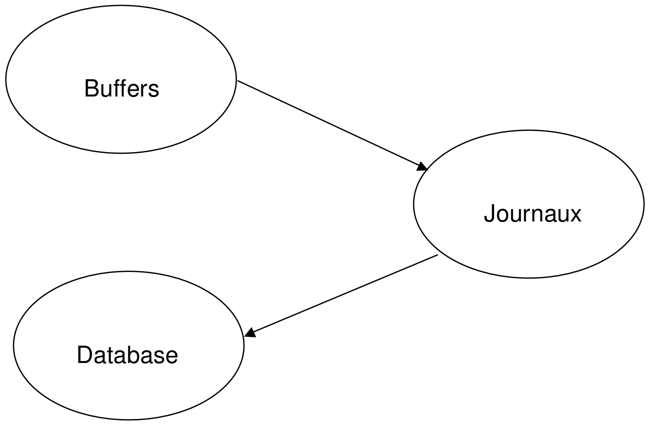
Partie 2: Fiabilité et Durabilité

Les différents types de « pannes » :

- **Sans perte d'information** : mémoire indemne
  ex : abandon d'une transaction (violation d'une CI, bug du programme utilisateur - div par 0)
- **Perte de la mémoire volatile**
  ex : panne système
- **Perte de la mémoire non volatile**
  ex : panne disque -> Duplication des données sur plusieurs disques
- **Perte de la mémoire permanente**
  plusieurs pannes de mémoire non volatile

12 mars 2001 **14**

Validation d'une transaction



12 mars 2001 15

Conserver un historique : les journaux

Technique de base pour résister aux pannes

On **enregistre** les informations concernant **toutes les opérations** de **toutes les transactions**

Permettent de défaire (**undo**) ou de refaire (**redo**) une transaction en cas de panne

- défaire les effets d'une transaction interrompue (avant validation)
- refaire les effets d'une transaction validée interrompue avant répercussion des effets
- undo et redo sont des opérations idempotentes (pannes récurrentes)

Contenu d'un fichier log (transaction id, record id, type of action - insert delete update-, old value, new value, info compl)

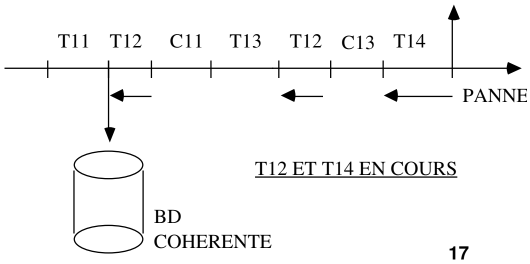
On enregistre le journal avant d'exécuter la mise à jour sur disque (**write ahead protocol**)

12 mars 2001 16

Procédures de recouvrement

- **Perte de mémoire volatile**
  - Déterminer toutes les transactions non validées (pas d'instruction de fin de transaction dans le log)
  - Déterminer toutes les transactions qui doivent être refaites
  - Défaire le premier ensemble de transactions et refaire le deuxième
  - **Checkpoints**
  - Contient à un instant donné la liste des transactions non validées
  - Permet de ne pas lire la totalité des fichiers journaux

- **Perte de mémoire stable**
  - Partir d'une sauvegarde de la base sur bande
  - refaire toutes les transactions validées du journal



12 mars 2001 17

Tout un choix d'algorithmes

Undo / No-redo

after image non nécessaire dans le log, Restart peu cher, abandon de transaction cher (=> algo de concurrence pessimiste)

No-undo / Redo

pas de before image dans le log, abort peu cher, contrôle de concurrence optimiste (abandon fréquent de transaction)

Undo / Redo

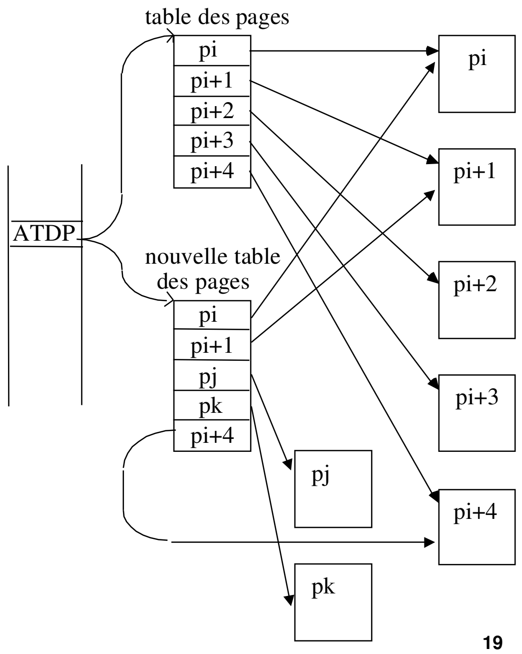
pages modifiées sauvegardées avant, pendant ou après le commit. obj : meilleur ordonnancement du gestionnaire de page. Contrôle de concurrence pessimiste. Technique utilisée par la plupart des systèmes.

No-undo / No-redo

Lié à une réalisation des maj par pages ombrées

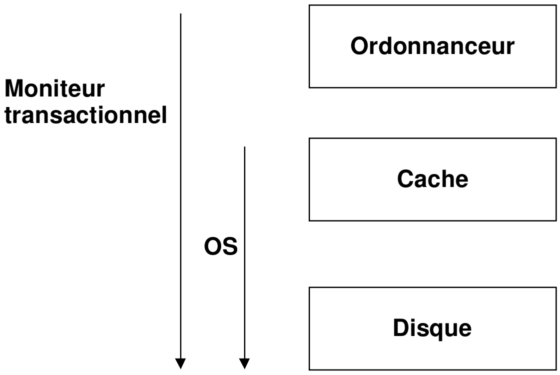
12 mars 2001 18

Principe des pages ombrées



12 mars 2001 19

Conclusion



12 mars 2001 20