

Ingénierie des Bases de Données

Fiabilité



Référence

Database systems

The complete book, chapitre 17

Garcia-Molina, Ullman, Widom



Fiabilité (Reliability/Recovery/Resilience)

Capacité du système SGBD à remédier aux erreurs et pannes

- Erreurs dans les programmes d'application (transactions)
- Erreurs dans l'entrée des données
- Erreurs d'enregistrement sur disques et crash matériels
- Catastrophes
- Défaillances système



Fiabilité

- Fiabilité= capacité à restaurer la base de données dans un état connu comme correct après une erreur quelconque qui a amené la base de données dans un état non correct.
- Objectifs:
 - ◆ éviter que la BD soit incohérente
 - ◆ éviter que les pg donnent des résultats faux
- Différents types de pannes/erreurs peuvent induire un état incorrect de la BD



Erreurs dans l'entrée des données

- Erreurs détectables :
 - ◆ par les contraintes d'intégrité
 - ◆ par les triggers
- Erreurs non détectables :
 - ◆ valeurs vraisemblables mais incorrectes (par exemple, année de naissance 1978 au lieu de 1987)



Erreurs disques

- Détection de l'erreur: Utilisation des bits de parité pour vérifier les enregistrements au niveau secteur
- Solutions proposées en cas de crash de la tête de lecture : Redondance
 - 1) RAID: "Redundant Arrays of Independent Disks"*
 - RAID niveau 1 : mirroring (doublement des disques)
 - RAID niveau 4 : un seul disque miroir avec les bits de parité de tous les disques et du disque miroir
 - RAID niveau 5 : le disque miroir est réparti
 - 2) Archivage
 - 3) Plusieurs BD réparties avec duplication (copies multiples)

* chapitre 11 du livre: Database systems, the complete book



Catastrophes

- Exemple:
 - ◆ Incendie, inondation, explosion, ...
 - ◆ → destruction des données
- Solution: **Redondance**
 - 1) Copies d'archive
 - 2) BD réparties avec duplication (copies multiples)

7 

Défaillance système

- **Défaillances système: pannes/erreurs qui entraînent la mauvaise exécution d'une transaction et donc BD incohérente**
- Exemples les plus courants:
 - ◆ Pannes (bugs, coupure courant) et erreurs de logiciel
- Problèmes:
 - ◆ Perte ou altération du contenu de la mémoire centrale
 - ◆ En particulier, perte de l'information sur les transactions en cours
- Solution:
 - ◆ Journalisation + Procédure de reprise après panne

8 

Programmes → Transactions

- Transaction = unité de programme exécutée sur SGBD
- Début Transaction
 - | Accès à la base de données (lectures, écritures)
 - | Calculs en mémoire centrale
- Fin Transaction : COMMIT ou ROLLBACK
- Commit : exécution correcte → validation
- Rollback : exécution incorrecte → effacement

9 

Gestion des transactions / fiabilité

- **Une transaction s'exécute correctement si:**
 - ◆ **Atomicité** (+ cohérence, isolation, durabilité)
 - tout ou rien
- Gestionnaire de la fiabilité doit s'assurer que cette atomicité est maintenue lors de pannes.

10 

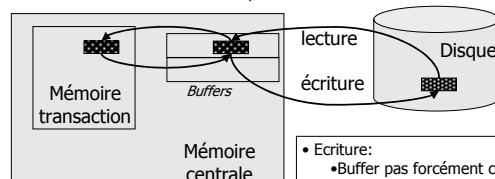
Transfert des données

- Le transfert de données transaction <-> disque de la BD transite par des tampons (buffers) :
 - ◆ gestionnaire des Buffers
- Unité de transfert en entrée/sortie: un bloc disque

11 

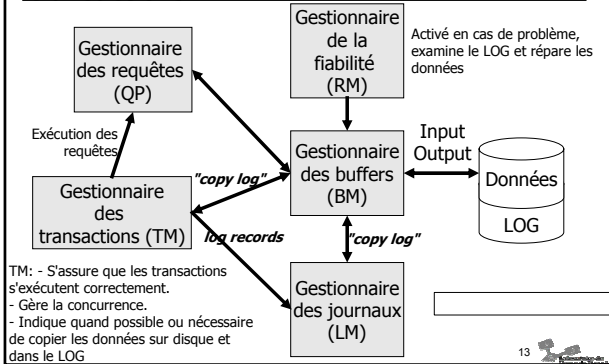
Interaction transactions - BD

- 3 espaces de stockage:
 - ◆ Disque (BD)
 - ◆ Zone de mémoire centrale (gérée par le gestionnaire des Buffers)
 - ◆ Zone de mémoire réservée pour la transaction



- Ecriture:
 - Buffer pas forcément copié sur disque tout de suite, peut rester en mémoire (dangereux), décidé par BM
 - Une des étapes du processus de fiabilité est de FORCER le BM d'écrire les données sur disque

Transfert de données: Gestionnaire des transactions



13

Operations primitives:

- Input (x): block contenant x → mémoire
 - Output (x): block contenant x → disque (et LM)
 - Read (x,t): faire input(x) si nécessaire
t ← valeur de x
 - Write (x,t): faire input(x) si nécessaire
t → x (en mémoire)
- Transactions

14

Exemple

Action de T	v	M.A	M.B	D.A	D.B
1) Read (A,v)	8	8		8	8
2) v := v*2	16	8		8	8
3) Write (A,v)	16	16		8	8
4) Read (B,v)	8	16	8	8	8
5) v := v*2	16	16	8	8	8
6) Write (B,v)	16	16	16	8	8
7) Output (A)	16	16	16	16	8
8) Output (B)	16	16	16	16	16

BM

Panne

15

Exemple - suite

- Que faire ?
 - Remettre A et B à valeur initiale: 8
 - Mettre B à 16
- Comment?

16

Solutions

- Journalisation (sur support non volatile) de l'activité des transactions : LOG
 - Objectif: s'assurer que du point de vue de la BD l'atomicité est respectée
 - Trois types de journalisation
 - UNDO
 - REDO
 - UNDO/REDO
- Procédure de reprise après panne
 - "Recovery": reconstruction de la BD à partir du LOG
 - "Checkpointing": limite la taille du LOG
 - Archivage: restauration de la BD même après grosses pannes

17

Le journal (log)

- Séquence d'enregistrements de LOG

The diagram shows a sequence of log records represented as blocks. T1 has 2 records, T2 has 2 records, T3 has 2 records, and T4 has 2 records. The records are arranged in a sequence, with some records from T1 and T2 appearing before the records of T3 and T4.
- Enregistre les actions de toutes les transactions, Append only
- Les enregistrements sont en mémoire centrale comme les données de la BD et sont écrits périodiquement sur disque.
- Commande FLUSH LOG permet de forcer la copie du contenu du Log de la mémoire centrale sur le disque
- En cas de panne, log est utilisé:
 - soit pour annuler des actions faites par transactions UNDO
 - soit pour refaire des actions faites par transactions REDO
 - soit pour annuler certaines actions et refaire d'autres UNDO/REDO

18

Le journal (log)

- Articles de différents types :
 - ◆ <Start T>
 - ◆ <Commit T>
 - ◆ <Abort T>
 - ◆ <T, X, v >
 - signifie: Transaction T a mis à jour élément X.
 - où X: élément mis à jour, v: ancienne valeur de X
- <T, X, v> est généré par un WRITE
 - (pas par OUTPUT donc pas encore forcément sur disque)

19 

Techniques de reprise (recovery)

- UNDO
 - ◆ défait l'action des transactions non validées*
- REDO (deferred update)
 - ◆ refait l'action des transactions non validées
- UNDO / REDO (immediate update)
 - ◆ défait ou refait suivant l'état de la transaction

*transaction validée= transaction ayant fait un Commit

20 

Règles de journalisation UNDO

- U 1) Écrire les mises à jour <T,X,v> dans le journal avant de modifier la BD (valeur de X) sur le disque.
- U 2) N'écrire le COMMIT dans le journal que après l'écriture sur le disque de toutes les mises à jour de la transaction
- Cet ordre s'applique individuellement pour chaque mise à jour

21 

Règles de journalisation UNDO (2)

- Signifie que on doit faire dans l'ordre suivant:
 - ◆ 1) écrire sur disque les enregistrements LOG indiquant les éléments BD changés
 - ◆ 2) écrire sur disque les éléments BD changés eux-mêmes
 - ◆ 3) écrire l'enregistrement COMMIT du LOG sur le disque

22 

Exemple de journalisation undo

Action de T	v	M.A	M.B	D.A	D.B	Journal
1						< Start T >
2 Read (A,v)	8	8		8	8	
3 v := v*2	16	8		8	8	
4 Write (A,v)	16	16		8	8	< T, A, 8 >
5 Read (B,v)	8	16	8	8	8	
6 v := v*2	16	16	8	8	8	
7 Write (B,v)	16	16	16	8	8	< T, B, 8 >
8 Flush Log	On écrit le LOG avant d'écrire les MAJ sur disque					
9 Output (A)	16	16	16	16	8	
10 Output (B)	16	16	16	16	16	
11						< Commit T >
12 Flush Log	On met l'enregistrement Commit après MAJ données sur disque					

23 

Procédure de reprise (undo)

- Objectif: restaurer l'atomicité
- Examiner le fichier log (en remontant du plus récent au moins récent):
 - ◆ Pour les transactions pour lesquelles existe un COMMIT sur disque
 - ne rien faire
 - ◆ Pour les autres transactions (on a <START T> mais pas de <COMMIT T>), pour chaque entrée < T, X, v>:
 - défaire chaque mise à jour: réécrire l'ancienne valeur
 - faire WRITE (X,v); OUTPUT(X);
 - écrire un <Abort T> dans le fichier log
 - Flush Log

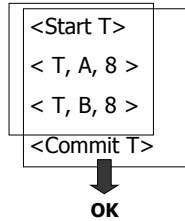
24 

Procédure de reprise - EXEMPLE

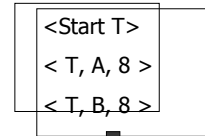
Action de T	Journal	
1	< Start T >	▪ si panne après 12: pas de pb
2	Read (A,v)	▪ si panne à 11 et <Commit T> non écrit dans le LOG, ou entre 8 et 11:
3	v := v*2	♦ défaire T:
4	Write (A,v)	▪ Write(B, 8); output(B)
5	Read (B,v)	▪ Write(A, 8); output(A)
6	v := v*2	▪ Ajouter <ABORT T> dans LOG
7	Write (B,v)	▪ FlushLog
8	Flush Log	▪ si panne avant 8,
9	Output (A)	♦ <T, B, 8> et <T,A,8> non écrits dans le LOG mais pas de problèmes car les données n'ont pas été écrites sur disque.
10	Output (B)	
11	< Commit T >	
12	Flush Log	

25

En résumé:



Si nouvelle panne pendant reprise, pas de pb car on réécrit l'ancienne valeur, on peut recommencer une procédure de Reprise



Write(B,8); Output(B)
Write(A,8); Output(A)
ajouter <Abort T> dans LOG
Flush Log

26

Checkpoint

- A priori en cas de reprise, on devrait remonter tout le fichier log.
 - ♦ Mais peut être long!
- Idée: « stabiliser » le log périodiquement
 - ♦ écriture d'un CHECKPOINT

27

Checkpoint

- Checkpoint:
 - 1) Ne plus accepter de nouvelles transactions
 - 2) Attendre la fin des transactions en cours et qu'elles aient écrit un enregistrement Commit ou Abort sur le LOG
 - 3) Flush Log
 - 4) Ecrire un enregistrement <CKPT>
 - 5) Flush Log
 - 6) Recommencer à accepter de nouvelles transactions

28

Reprise avec Checkpoint

- Examiner le fichier log (en remontant) jusqu'à l'enregistrement <CKPT>
- Les transactions avant le CKPT se sont terminées correctement et leurs mises à jour ont été écrites sur disque.
- Les transactions après le CKPT doivent être défaites comme précédemment.

29

Exemple

```

<START T1>
<T1, A, 5>
<START T2>
<T2, B, 10>
<T2, C, 15>
<T1, D, 20>
<COMMIT T1>
<COMMIT T2>
<CKPT>
<START T3>
<T3, E, 25>
<T3, F, 30>
----- PANNE
    
```

On décide de faire un CKPT

On attend la fin de T1 et T2 avant d'écrire <CKPT>

T3 est la seule transaction incomplète qui doit faire l'objet d'une reprise

30

Nonquescent Checkpoint

- Objectif : ne pas ralentir le système en refusant de nouvelles transactions
- Méthode Nonquescent Checkpoint :
 - 1) Enregistrer dans le fichier log un enregistrement contenant la liste des transactions en cours:
< START CKPT (T1, T2, ... Tn) >
 - 2) Flush Log
 - 3) Attendre la fin des transactions en cours, sans interdire le démarrage de nouvelles transactions
 - 4) Quand les transactions T1, T2, ... Tn ont terminé, écrire un enregistrement < END CKPT >
 - 5) Flush Log

31

Reprise avec Nonquescent Checkpoint

- Reprise :
 - Si on rencontre un <END CKPT> alors on sait que toutes les transactions incomplètes sont situées après le dernier < START CKPT(...) > donc pas la peine de remonter plus en avant que ce START CKPT.
 - Si on rencontre un < START CKPT(T1, ...Tn)> alors la panne a eu lieu pendant le checkpoint. Les transactions incomplètes sont celles rencontrées entre la panne et le < START CKPT(...) > et celles parmi T1, ..., Tn qui ne sont pas terminées. On doit remonter jusqu'au <START > de la plus vieille de ces transactions.

32

Exemple (1)

LOG <START T1> <T1, A, 5> <START T2> <T2, B, 10> <START CKPT (T1,T2)> <T2, C, 15> <START T3> <T1, D, 20> <COMMIT T1> <T3, E, 25> <COMMIT T2> <END CKPT> <T3, F, 30> ----- PANNE	<T3, F, 30> - T3 doit être défaite car on ne trouve pas de COMMIT -> on réécrit la valeur 30 pour F <END CKPT> - on sait que toutes les transactions incomplètes sont situées après le 1er <START CKPT> trouvé. <T3, E, 25> - on doit réécrire la valeur 25 pour E <T1, D, 20> - comme on a rencontré un <COMMIT T1>, T1 s'est bien terminée donc on ne fait rien <T2, C, 15> - comme on a rencontré un <COMMIT T2>, T2 s'est bien terminée donc on ne fait rien <START CKPT (T1,T2)> - on s'arrête
--	---

33

Exemple (2)

LOG <START T1> <T1, A, 5> <START T2> <T2, B, 10> <START CKPT (T1,T2)> <T2, C, 15> <START T3> <T1, D, 20> <COMMIT T1> <T3, E, 25> ----- PANNE	<T3, E, 25> - Pas de Commit donc T3 s'est mal terminée, on doit la défaire (réécrire 25 pour E) <T1, D, 20> - comme on a rencontré un <COMMIT T1>, T1 s'est bien terminée donc on ne fait rien <START T3>: T3 a été complètement défaite <T2, C, 15> - Pas de Commit donc T2 s'est mal terminée, on doit la défaire (réécrit C=15) <START CKPT (T1,T2)> - Les transactions incomplètes sont celles rencontrées entre la panne et le START CKPT(T1,...Tn) et celles parmi T1, ..., Tn qui ne sont pas terminées: (T1, T2, T3) - Or T1 a été validée, T3 défaite. - On doit continuer jusqu'au START de T2 (seule transaction qui reste n'ayant pas été entièrement défaite) <T2, B, 10>: - on réécrit B=10 <START T2>: - On s'arrête
--	---

34

Règles de journalisation REDO (1)

- Inconvénient de Journalisation UNDO: on ne peut pas faire un commit d'une transaction sans avoir écrit toutes les données modifiées sur le disque (output).
- Or il peut être intéressant (gain I/O) de laisser les données en mémoire avant d'aller les écrire sur disque
- Journalisation REDO: permet de ne pas écrire immédiatement les mises à jour sur les disques

35

Règles de journalisation REDO (2)

- UNDO: défaite les transactions incomplètes et ignore les transactions validées
- REDO: ignore les transactions incomplètes et refait les transactions validées
- UNDO : écrit les mises à jour avant d'écrire le Commit
- REDO : écrit le Commit avant d'écrire les mises à jour
- UNDO : log des mises à jour avec anciennes valeurs
- REDO : log des mises à jour avec nouvelles valeurs

36

Règles de journalisation REDO (3)

- Mêmes enregistrements que pour UNDO
- Sauf signification de $\langle T, X, v \rangle$
 - ◆ Transaction T écrit la nouvelle valeur v pour X
- Règle d'écriture dans le fichier LOG:

R1) Avant de modifier un élément de la BD sur le disque, tous les enregistrements de LOG liés à cette modification ($\langle T, X, v \rangle$ et $\langle \text{COMMIT } T \rangle$) doivent être sauvegardés sur le disque

37

Exemple de journalisation redo

Action de T	v	M.A	M.B	D.A	D.B	Journal
1) READ (A,v)	8	8		8	8	$\langle \text{Start } T \rangle$
3) $v := v * 2$	16	8		8	8	
4) WRITE (A,v)	16	16		8	8	$\langle T, A, 16 \rangle$
5) READ (B,v)	8	16	8	8	8	
6) $v := v * 2$	16	16	8	8	8	
7) WRITE (B,v)	16	16	16	8	8	$\langle T, B, 16 \rangle$
8)						$\langle \text{Commit } T \rangle$
9) Flush Log						← On écrit le LOG avant d'avoir écrit les données sur disque
10) Output (A)	16	16	16	16	8	
11) Output (B)	16	16	16	16	16	

On met l'enregistrement Commit AVANT MAJ données sur disque

38

Règle pour redo

- Avant de modifier une donnée sur le disque, écrire le $\langle T, X, v \rangle$ et le $\langle \text{Commit } T \rangle$ dans le LOG
- Reprise :
 - 1) Identifier les transactions qui ont fait Commit
 - 2) Balayer le fichier LOG du début. Pour chaque ordre $\langle T, X, v \rangle$:
 - Si T n'a pas fait de Commit: ignorer la mise à jour (elle n'a pas été écrite sur le disque),
 - Si T a fait un Commit: refaire la mise à jour (écrire la valeur v pour X)
 - 3) Pour chaque transaction incomplète T, écrire $\langle \text{Abort } T \rangle$, Flush Log

39

Exemple reprise pour REDO

Action de T	Journal
1) READ (A,v)	$\langle \text{Start } T \rangle$
2) $v := v * 2$	
4) WRITE (A,v)	$\langle T, A, 16 \rangle$
5) READ (B,v)	
6) $v := v * 2$	
7) WRITE (B,v)	$\langle T, B, 16 \rangle$
8)	$\langle \text{Commit } T \rangle$
9) Flush Log	
10) Output (A)	
11) Output (B)	

- si panne après 9: LOG contient l'enregistrement $\langle \text{Commit } T \rangle$. On va donc réécrire les valeurs pour A et B telles que dans les enregistrements $\langle T, A, 16 \rangle$ et $\langle T, B, 16 \rangle$
- si panne entre 10 et 11 On réécrit les valeurs de A et B, la valeur de A est déjà écrite mais ce n'est pas gênant
- si panne avant 8:
 - ◆ pas de Commit dans le LOG. T est considérée comme une transaction incomplète
 - ◆ Les MAJ de T n'ont pas été écrites sur disque, on ne fait rien
 - ◆ Ajouter $\langle \text{ABORT } T \rangle$ dans LOG
- si panne entre 8 et 9:
 - ◆ si commit écrit (flush par autre transaction par ex), idem après 9
 - ◆ sinon idem avant

Checkpoint

- Vu que le commit est écrit avant le output, les MAJ faites par une transaction peuvent avoir été copiées sur disque bien après que le COMMIT ait eu lieu
- Donc le CKPT ne peut pas regarder que les transactions actives qui sont validées ou non.
- $\forall$ Quiescent ou nonquiescent: durant le CKPT on doit forcer l'écriture sur disque des éléments modifiés par des transactions validées.

41

Checkpoint et redo

- Étapes:
 - 1) Écrire $\langle \text{START CKPT } (T_1, T_2, \dots, T_n) \rangle$ où $T_1, \dots, T_n$ sont les transactions actives non validées (pas Commit)
 - 2) Flush Log
 - 3) Écrire sur le disque (vider les tampons) les mises à jour des transactions validées (commit) avant le checkpoint
 - 4) Écrire $\langle \text{END CKPT} \rangle$
 - 5) Flush Log

42

Reprise avec Checkpoint (1)

- Si la panne a lieu après <END CKPT>
 - On sait que les valeurs écrites par transactions validées (Commit) avant <START CKPT(T1,...,Tn)> sont maintenant sur disque.
 - Les transactions parmi T1, ..., Tn ou ayant commencé après le <START CKPT>, n'ont pas forcément leurs valeurs écrites sur disque même si elles ont fait Commit.
 - On doit donc faire la reprise pour REDO de ces transactions:
 - Si T n'a pas fait de Commit: ignorer la mise à jour (elle n'a pas été écrite sur le disque),
 - Si T a fait un Commit: refaire la mise à jour (écrire la valeur v pour X)
 - On ne doit pas regarder plus haut dans le LOG que le plus vieux <START Ti> où Ti est une de T1, ..., Tn ou ayant commencé après le <START CKPT>

43

Reprise avec Checkpoint (2)

- Si la panne a lieu après un < START CKPT(T1, ..., Tn)>
 - On ne sait pas si les valeurs écrites par transactions validées (Commit) avant <START CKPT(T1,...,Tn)> sont sur disque.
 - On doit trouver le dernier enregistrement <END CKPT>, trouver son <START CKPT(S1,...,Sn)> et refaire toutes les transactions validées qui ont soit commencé après ce <START CKPT(S1,...,Sn)> ou qui font partie de S1...Sn

44

Reprise avec Checkpoint

LOG

```
<START T1>
<T1, A, 5>
<START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT (T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>
----- PANNE
```

- On recherche le <END CKPT>
- Les transactions qu'on va refaire sont soit celles qui ont commencé après <START CKPT (T2)> ou {T2}
 - {T2, T3}
- Les valeurs mises à jour par T1 sont sur disque
- T2 et T3 ont fait un Commit, leurs valeurs doivent être réécrites
- On remonte jusqu'à <START T2> (la plus ancienne), on trouve les MAJ:
 - <T2, B, 10>
 - <T2, C, 15>
 - <T3, D, 20>
- On réécrit ces 3 valeurs

45

Reprise avec Checkpoint

LOG

```
<START T1>
<T1, A, 5>
<START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT (T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>
<END CKPT>
<COMMIT T2>
----- PANNE
<COMMIT T3>
```

- On recherche le <END CKPT>
- Les transactions qu'on va refaire sont soit celles qui ont commencé après <START CKPT (T2)> ou {T2}
 - {T2, T3}
- Les valeurs mises à jour par T1 sont sur disque
- Seule T2 a fait un Commit, ses valeurs doivent être réécrites. Les MAJ de T3 sont ignorées
- On remonte jusqu'à <START T2> (la plus ancienne), on trouve les MAJ:
 - <T2, B, 10>
 - <T2, C, 15>
- On réécrit ces 2 valeurs
- On écrit un <ABORT T3>

46

Reprise avec Checkpoint

```
<START T1>
<T1, A, 5>
<START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT (T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>
----- PANNE
<END CKPT>
<COMMIT T2>
<COMMIT T3>
```

- On recherche le premier <START CKPT()> avant <START CKPT(T2)>
- Or il n'existe pas donc on remonte au début du LOG
- La seule transaction validée est T1.
- On refait sa mise à jour.
 - on réécrit la valeur 5 pour A
- On écrit un <ABORT T2> et <ABORT T3>

47

UNDO/REDO

- Inconvénients de UNDO et de REDO:
 - UNDO nécessite que les données soient écrites sur disque avant que le Commit soit écrit dans le journal ce qui entraîne bcp de I/O
 - REDO au contraire demande à ce que les données restent dans les buffers tant que la transaction n'est pas validée et que le fichier LOG est écrit. Le nombre de buffers nécessaires est donc augmenté.
- > Journalisation UNDO/REDO

48

UNDO/REDO

- Mêmes enregistrements que pour UNDO et REDO
- sauf UPDATE: $\langle T, X, v, w \rangle$
 - Transaction T a changé la valeur de X, l'ancienne valeur était v, la nouvelle valeur w
- Règle d'écriture dans le fichier LOG

UR1) Avant de modifier un élément X de la BD sur le disque, il est nécessaire qu'un enregistrement de mise à jour $\langle T, X, v, w \rangle$ soit écrit sur le disque.

$\langle COMMIT T \rangle$ peut être avant ou après l'écriture sur disque de X

49

Exemple de journalisation UNDO/REDO

Action de T	v	M.A	M.B	D.A	D.B	Journal
1) READ (A,v)	8	8		8	8	< Start T >
3) $v := v * 2$	16	8		8	8	
4) WRITE (A,v)	16	16		8	8	< T, A, 8, 16 >
5) READ (B,v)	8	16	8	8	8	
6) $v := v * 2$	16	16	8	8	8	
7) WRITE (B,v)	16	16	16	8	8	< T, B, 8, 16 >
8) Flush Log						< Commit T >
9) Output (A)	16	16	16	16	8	< Commit T >
10)						< Commit T >
11) Output (B)	16	16	16	16	16	< Commit T >

50

Reprise avec UNDO/REDO

- On a les informations soit pour défaire soit pour refaire une maj
- Reprise:
 - Refaire toutes les transactions validées (en descendant le log)
 - Défaire les transactions non-validées (en remontant le log)

51

Exemple de panne avec UNDO/REDO

Action de T	Journal
1) READ (A,v)	< Start T >
3) $v := v * 2$	
4) WRITE (A,v)	< T, A, 8, 16 >
5) READ (B,v)	
6) $v := v * 2$	
7) WRITE (B,v)	< T, B, 8, 16 >
8) Flush Log	
9) Output (A)	
10)	< Commit T >
11) Output (B)	

- Panne après 10 (mais journal comprend commit), T est validée. On refait toutes ses MAJ: on réécrit la nouvelle valeur de A (inutile) et B
- Panne avant 10: T n'est pas validée. Ses MAJ sont défaites: on réécrit l'ancienne valeur de A et B

52

Checkpoint avec UNDO/REDO

- Étapes:
 - Écrire $\langle \text{START CKPT } (T1, T2, \dots, Tn) \rangle$ où $T1, \dots, Tn$ sont les transactions actives (pas Commit)
 - Flush Log
 - Écrire sur le disque (vider les tampons) les mises à jour de toutes les transactions actives (validées ou non) ou terminées avant le checkpoint
 - Écrire $\langle \text{END CKPT} \rangle$
 - Flush Log

53

Reprise avec Checkpoint

LOG	
1) $\langle \text{START } T1 \rangle$	
2) $\langle T1, A, 4, 5 \rangle$	
3) $\langle \text{START } T2 \rangle$	
4) $\langle \text{COMMIT } T1 \rangle$	
5) $\langle T2, B, 9, 10 \rangle$	
6) $\langle \text{START CKPT } (T2) \rangle$	
7) $\langle T2, C, 14, 15 \rangle$	
8) $\langle \text{START } T3 \rangle$	
9) $\langle T3, D, 19, 20 \rangle$	
10) $\langle \text{END CKPT} \rangle$	
11) $\langle \text{COMMIT } T2 \rangle$	
12) $\langle \text{COMMIT } T3 \rangle$	
13)	

- T2 est la seule transaction active au moment du Checkpoint.
- Durant le Checkpoint, on écrit sur disque les valeurs de A et B
- Si une panne arrive en 13:
 - Pour T1 on ne fait rien, elle est validée et ses valeurs ont été écrites sur disque durant le CKPT
 - T2 et T3 ont été validées (Commit), on refait leurs mises à jour depuis le $\langle \text{START CKPT } (T2) \rangle$ car les mises à jour avant le CKPT ont déjà été écrites sur disque ($\langle T2, B, 9, 10 \rangle$)
- Si une panne arrive entre 11 et 12:
 - On refait T2, on défait T3

54

Comment reconstruire la BD après destruction du disque?

- Comment reconstruire la base de données après la destruction des disques contenant les données?
- Log:
 - ◆ restaurer les mises à jour après panne quand données en mémoire sont perdues mais les données sur disque ne sont pas perdues.
- Utilisation du LOG pour reconstruire la BD?
 - ◆ Il faudrait conserver tout le LOG
 - ◆ LOG conservé sur un autre disque que les données
 - ◆ LOG UNDO/REDO ou REDO (pour avoir les nouvelles valeurs)
 - ◆ Pas réaliste car LOG devient vite très volumineux

55 

Archivage

- Solution:
 - ◆ Archiver la base de données à date t
 - ◆ Reconstruire la BD en utilisant cette archive et éventuellement le LOG depuis le moment de l'archivage pour avoir un état plus avancé (t+x)
- Différents niveaux d'archivage:
 - ◆ "Full dump"
 - ◆ "Incremental dump"
 - ◆ Mixte:
 - niveau 0: "full dump"
 - niveau 1: "incremental dump" depuis niveau 0
 - niveau 2: "incremental dump" depuis niveau 1
 - ...

56 

Non-quiescent archive

- On ne peut pas arrêter la BD pour faire archivage
 - > non-quiescent archive
- Cependant:
 - ◆ La copie de la BD est faite alors que des mises à jour peuvent encore être faites sur la BD
 - ◆ Pour restaurer on a besoin de l'archive et du log des transactions actives pendant l'archivage
 - ◆ LOG UNDO/REDO ou REDO

57 

Exemple

- BD avec 4 éléments A, B, C, D
- État de la BD au début de l'archivage:
 - ◆ A=1, B=2, C=3, D=4
- État de la BD à la fin de l'archivage !
 - ◆ A=5, B=7, C=6, D=4
- État de l'archive !
 - ◆ A=1, B=2, C=6, D=4

Séquence d'évènements arrivant sur la BD:

Disque	Archive
A:=5	Copy A
C:=6	Copy B
B:=7	Copy C
	Copy D

Non-quiescent archive

- Méthode de création d'une archive:
 - 1) Écrire un enregistrement <START DUMP> sur le LOG
 - 2) Faire un checkpoint adapté à la méthode de journalisation utilisée (UNDO/REDO ou REDO)
 - 3) Faire une copie (full dump ou incremental dump)
 - 4) S'assurer que le fichier log à partir au moins du checkpoint (inclus) est sauvegardé sur un disque sûr
 - 5) Écrire un enregistrement <END DUMP> sur le LOG

59 

Exemple de fichier Log avec UNDO/REDO

LOG

```

<START DUMP>
<START CKPT (T1, T2)>
<T1, A, 1, 5>
<T2, C, 3, 6>
<COMMIT T2>
<T1, B, 2, 7>
<COMMIT T1>
<END CKPT>
----Fin de l'archivage
<END DUMP>

```

60 

Reprise avec archive et log

- Étapes:
 - ◆ Reconstruire la BD à partir de l'archive la plus récente
 - ◆ Modifier la BD à l'aide du LOG (utiliser la méthode de reprise correspondant au type de journalisation utilisée (UNDO/REDO ou REDO))

61



Exemple

- Exemple de BD avec A, B, C, D
 - 1) on restaure avec les valeurs de l'archive:
 - ◆ A=1, B=2, C=6, D=4
 - 2) on regarde le fichier LOG
 - ◆ Comme T1 et T2 ont été validées (Commit), on refait leurs MAJ:
 - <T1, A, 1, 5>, A:=5
 - <T2, C, 3, 6>, C:=6
 - <T1, B, 2, 7>, B:=7
- La BD restaurée a les valeurs suivantes:
 - ◆ A=5, B=7, C=6, D=4 **OK!**

62

