

Gestion des transactions

Plan

- ♦ I- Introduction aux transactions
 - 1. Notion de transaction
 - 2. Propriétés ACID
 - 3. Points de repère
 - 4. Etats de transactions
- ♦ II- Théorie de la concurrence
 - 1.Introduction: la concurrence de transactions
 - 2. Problèmes des accès concurrents:
 - 3. Sérialisation
 - 4.Niveaux d'isolation sous SQL
 - 5. Récupérabilité & Reprise sur panne
 - 6. Techniques de contrôle de concurrence
 - Verrouillage 2 phases,
 - Estampillage

II-THÉORIE DE LA CONCURRENCE

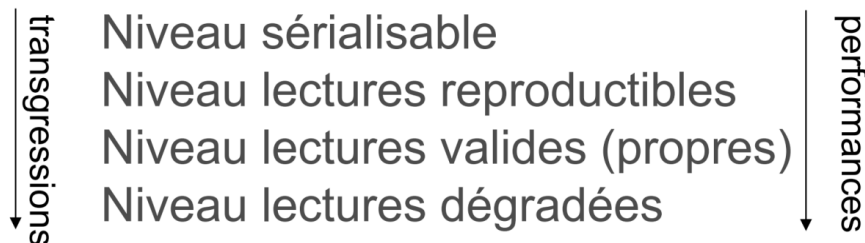
4- Niveaux d'isolation

Introduction

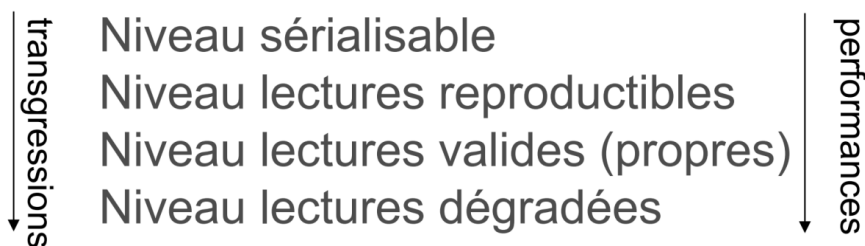
- ♦ La sérialisation est:
 - une propriété forte,
 - l'inconvénient est d'impliquer un contrôle strict du SGBD
 - Risque de pénaliser fortement les autres utilisateurs.
- ♦ Les systèmes proposent en fait plusieurs niveaux d'isolation dont chacun représente: un compromis entre:
 - la sérialisabilité, totalement saine mais pénalisante, et
 - une isolation partielle entraînant moins de blocages mais plus de risques d'interactions perturbatrices.

Niveaux d'isolation

- ◆ Pourquoi des anomalies peuvent-elles apparaître dans une exécution concurrente?
 - Parce que le niveau d'isolation n'est que partiel
- ◆ **Il existe plusieurs niveaux d'isolation: du plus permissif au plus strict.** (Définis en fonction des anomalies : lectures impropres, lectures non répétables, tuples fantômes.)
 - Plus le niveau est permissif, plus l'exécution est fluide, plus les anomalies sont possibles.
 - Plus le niveau est strict, plus l'exécution risque de rencontrer des blocages, moins les anomalies sont possibles.



Niveaux d'isolation



- ◆ Le niveau d'isolation totale, telle que défini ci-dessus, est dit *sérialisable* puisqu'il est équivalent du point de vue du résultat obtenu, à une exécution *en série* des transactions.
- ◆ Le choix du bon niveau d'isolation, pour une transaction donnée:
 - Est de la responsabilité du programmeur et
 - Implique une bonne compréhension des dangers courus et des options proposées par les SGBD.
- ◆ **Le niveau d'isolation par défaut n'est jamais le plus strict.** Car:
 - Inutile dans la grande majorité des cas ;
 - Provoque rejets et blocages difficiles à expliquer à l'utilisateur.
- ◆ **Quand l'isolation totale est nécessaire, il faut l'indiquer explicitement.**

4.1. Read Uncommitted

- ◆ Correspond à l'absence de contrôle de concurrence.
 - tout est permis, toutes les anomalies sont possibles.

SELECT SAL FROM emp WHERE NE = 1 → 2050	SELECT SUM(SAL) FROM EMP →5750	UPDATE emp SET SAL = 2100 WHERE NE = 1	SELECT SUM(SAL) FROM EMP →5800	SELECT SAL FROM EMP WHERE NE = 1 → 2100	Rollback	SELECT SAL FROM EMP WHERE NE = 1 → 2050
---	--	---	--	---	----------	---

T1: requêtes impropres

T1: lecture impropre car non
validée / lecture non reproductible

4.2. Read committed

- ◆ Les écritures bloquent les lectures (on ne peut lire que les nuplets validés)
 - Lectures « propres » (validées)
 - Lectures et requêtes non reproductibles
 - Apparition de fantômes et changement de valeurs déjà lues



SELECT SAL FROM EMP WHERE NE = 1 → 2050	SELECT SUM(SAL) FROM EMP →5750	UPDATE EMP SET SAL = 2100 WHERE NE = 1	COMMIT	SELECT SUM(SAL) FROM EMP →5800	SELECT SAL FROM EMP WHERE NE = 1 → 2100
---	---	--	--------	---	---

T2: l'écriture bloque
toute lecture jusqu'à
commit

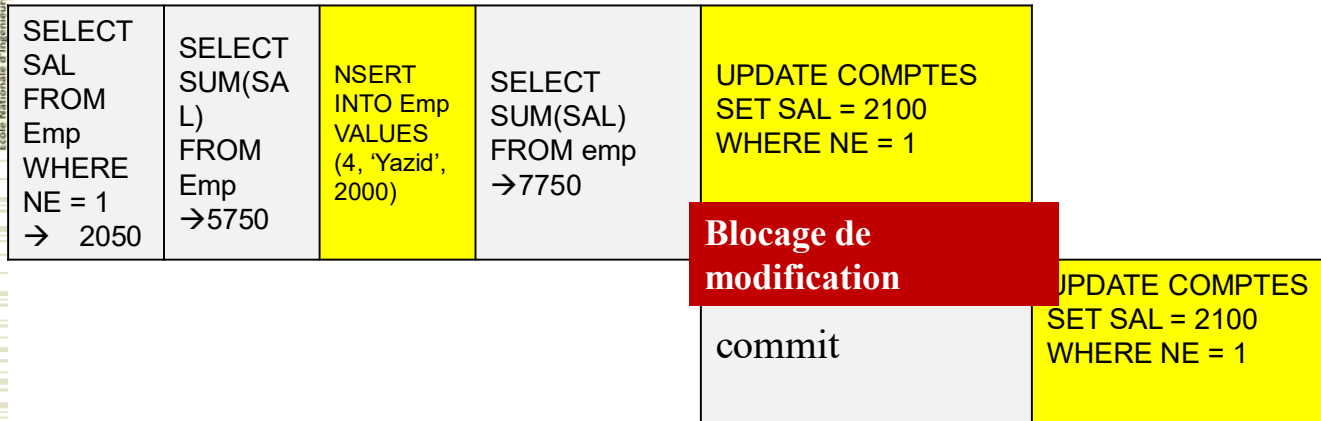
T2: Ce commit permet
à T1 de lire des valeurs
propres

T1: **requête** non
reproductible: changem
ent de valeurs déjà lues

T1: lecture propre mais
pas reproductible

4.3.Repeatable Read

- : les lectures aussi bloquent les écritures (seules les modif): une écriture qui modifie la valeur lue doit être retardée après la lecture et la fin de la transaction
 - seuls les tuples fantômes sont permis.
 - Lectures reproductibles, mais requêtes non reproductibles
 - Apparition de fantômes (Phantom Read = nouveaux tuples)



4.4.Serializable

- Serializable : isolation totale, aucune anomalie, interblocages possibles.
 - La sérialisabilité complète des transactions est assurée
 - Les données auxquelles on accède sont verrouillées jusqu'à la fin de la transaction
- ♦ Niveau par défaut : read committed (Oracle) ou repeatable read (MySQL, PostgreSQL)

Exemple récapitulatif

- ♦ Deux transactions sur un granule x (au début $x = 1$)
 - T1 : R1 [x] R1 [x] c1 - deux consultations successives de x
 - T2 : R2 [x] W2 [x] W2 [x] c2 - ajoute 1 à x à deux reprises
 - Séquence reçue: R1 [x] R2 [x] W2 [x] R1 [x] W2 [x] c2 c1
- ♦ Read uncommitted
 - R1 [x](1) R2 [x](1) W2 [x](2) R1 [x](2) W2 [x](3) c2 c1
 - Pas de retard, lectures impropre (lecture d'une valeur modifiée pas encore confirmée)
 - Peut être acceptable si T1 n'écrit pas
- ♦ Read committed
 - R1 [x](1) R2 [x](1) W2 [x](2) **W2 [x](3) c2 R1 [x](3) c1**
 - Évite lecture impropre
 - lecture retardée après la validation de l'écriture
 - N'évite pas la lecture non reproductible : lecture de deux valeurs différentes au sein de T1
- ♦ Repeatable read
 - R1 [x](1) R2 [x](1) **R1 [x](1) c1 W2 [x](2)** W2 [x](3) c2
 - En plus: écritures des autres transactions retardées après la fin de T1
- ♦ Serializable – En plus: insertions/suppressions des autres transactions retardées après T1

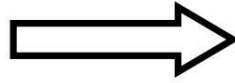
Bilan

- ♦ Degrés d'isolation Ansi SQL-92

Niveau d'Isolation	Lectures impropre	Lectures non reproductibles	Lectures fantômes
Read Uncommitted	Possibles	Possibles	Possibles
Read Committed	☒☺	Possibles	Possibles
Repeatable Read	☒☺	☒☺	Possibles
Serializable	☒☺	☒☺	☒☺

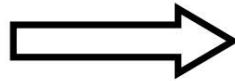
Choix du niveau d'isolation

- Beaucoup de lectures
- Peu ou pas d'écritures
- Transactions longues
- Peu de transactions



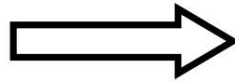
READ COMMITTED
Mode par défaut Oracle

- Peu de lectures
- Peu d'écritures
- Transaction courtes
- Beaucoup de transactions



SERIALIZABLE
REPEATABLE READ
Mode par défaut MySQL

- Systèmes d'inspection des données (debug)



DIRTY READ

II-THÉORIE DE LA CONCURRENCE

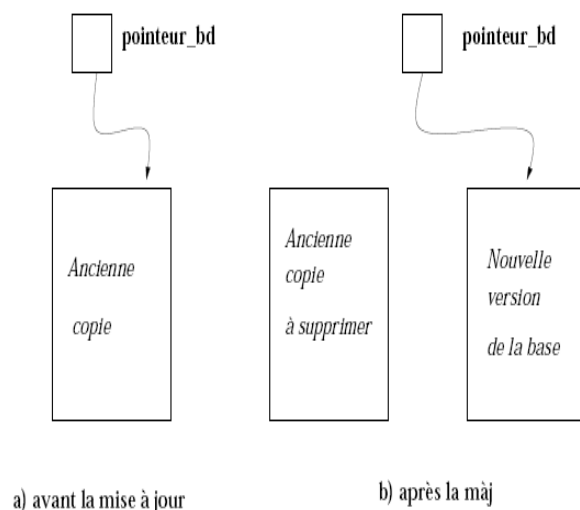
5- La récupérabilité & Reprise sur panne

5.1. Objectif de la reprise

- ♦ Types de pannes
 - *De transaction*: annulation d'une transaction
 - *De système*: perte du contenu de la mémoire vive
 - *De support*: perte du contenu du disque
- ♦ Au moment de la panne, l'état de la BD peut être incohérent
- ♦ Objectif
 - → **ramener la BD à son état validé avant la panne**
- ♦ Comment?
 - *Assurer l'atomicité*: annuler dans la BD les transactions non validées au moment de la panne
 - *Assurer la durabilité*: reproduire dans la BD l'effet des transactions validées au moment de la panne → Pour assurer la reprise → *journal* à stocker sur disque

5.2. Implémentation de l'Atomicité et journalisation

- ♦ C'est le mécanisme de reprise sur panne. Il emploie:
- ♦ La notion de copie (shadow database)
 - On suppose qu'une seule transaction peut être exécutée à la fois
 - Un pointeur `pointeur_bd` pointe vers la version cohérente courante de la base.
 - Toutes les mises à jour sont exécutées sur une copie. `Pointeur_bd` ne pointera sur la copie que si la transaction est validée.
 - Si la transaction échoue, alors la copie est supprimée.
- ♦ Journalisation (cas d'oracle)
 - Journal d'images après (redo log)
 - Journal d'images avant (undo segment)



5.3. Récupérabilité

◆ Principe:

- O est récupérable si suite à l'annulation d'une transaction, on peut toujours revenir à un état cohérent **sans défaire les transactions validées**.

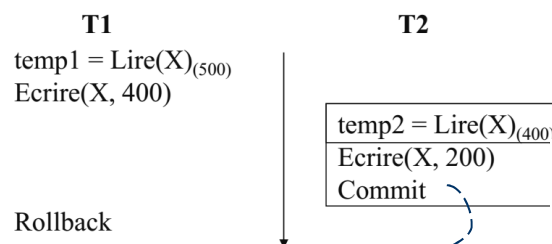
◆ Définition:

- Un ordonnancement O1 est récupérable : ssi: étant donné une transaction T_i qui lit un objet précédemment écrit par une deuxième transaction T_j , la validation de T_j a eu lieu avant la validation de T_i .
- On imposera alors que les exécutions concurrentes satisfassent les propriétés suivantes :
 - Recouvrabilité,
 - éviter les annulations en cascade,
 - exécution stricte.

5.3.1. Recouvrabilité

◆ Question: quand risque-t-on d'annuler une transaction déjà validée?

◆ Exemple:

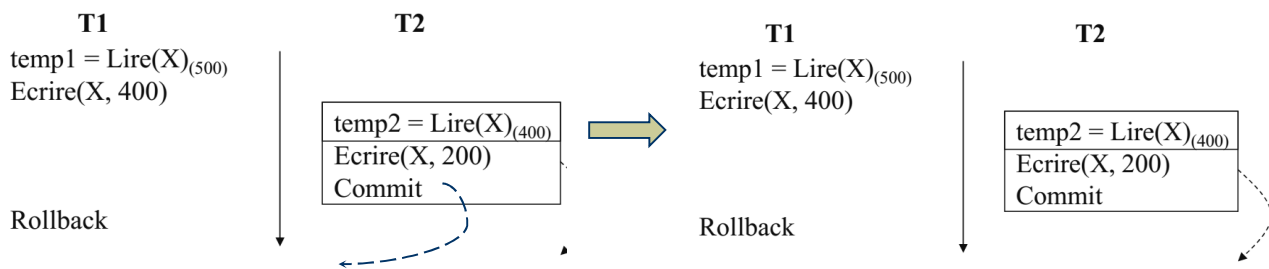


- Réponse: quand la transaction qui fait la lecture impropre est validée avant la fin de celle qui a fait l'écriture

- ◆ Le rollback de T_1 oblige l'annulation de T_2 . **Or, ce n'est pas possible car T_2 a été validée. On parle d'exécution non recouvrable.**
- ◆ L'ordonnancement suivant n'est pas récupérable car T_2 valide tout de suite après la lecture. $\Leftrightarrow T_1$ a été avortée, : T_2 a lu une valeur qui peut être "impropre"
- ◆ Solution pour éviter les transactions non recouvrables :
 - si T_2 lit au moins un enregistrement dont T_1 est la dernière transaction à l'avoir mis à jour, alors T_2 doit valider après $T_1 \rightarrow$ **retardement des validations**

5.3.2. Annulations en cascade

Annulations en cascade



- ◆ Même si le commit de T2 a été retardé, T2 sera quand même annulée à cause de l'annulation de T1: il s'agit d'une annulation en cascade.
- ◆ Solution : T2 ne doit lire qu'à partir de transactions validées
→ retardement des lectures jusqu'à validation de T1

5.3.2 Annulations en cascade

- ◆ Ordonnancement sans cascade :
 - Si pour chaque paire T_i, T_j / T_j lit une donnée précédemment écrite par T_i, alors la validation de T_i a lieu avant la lecture faite par T_j.
- ◆ Un ordonnancement sans cascade de rollback est récupérable.
- ◆ Il est souhaitable de restreindre les ordonnancements à ceux qui sont sans cascade

5.3.3.Exécution stricte

- ♦ Cas d'exécution stricte:

T_{10}	T_{11}
$Lire(A)$	
$Lire(B)$	
$Ecrire(A)$	
	$Lire(A)$
	$Ecrire(A)$
fin	
	fin

- ♦ Solution : écrire (A) (par T_{11}) attend que tout T_i qui a écrit A se termine (par a1 ou par c1), donc retardement de écrire(A) par T_{11} après la fin de T_{10} .
- ♦ → retardement des lectures et des écritures

II-THÉORIE DE LA CONCURRENCE

6- Techniques de Contrôle de concurrence

Introduction

- ♦ L'objectif du contrôle de concurrence: produire des exécutions sérialisables
 - Théorème de sérialisabilité : insuffisant, permet seulement de savoir si une exécution est sérialisable ou non
 - Algorithmes de contrôle de concurrence
 - Pas besoin de graphe de précédence
 - Modifient l'entrelacement des transactions pour rendre l'exécution sérialisable
 - Réordonnancement des opérations
 - Ordonnanceur ("scheduler"): module logiciel qui exécute l'algorithme de contrôle de concurrence
 - Le SGBD reçoit une séquence d'exécution d'entrée que l'ordonnanceur modifie pour rendre l'exécution concurrente correcte
- ♦ Propriétés d'annulation (Recouvrabilité)
 - Les propriétés d'annulation sont orthogonales à la sérialisabilité
 - On peut avoir toutes les combinaisons: sérialisable mais pas recouvrable, stricte mais pas sérialisable, etc.
 - L'ordonnanceur essaie d'assurer en plus de la sérialisabilité la meilleure propriété d'annulation possible dans le cadre de son algorithme

1. Algorithmes de contrôle de concurrence

- ♦ Deux types de techniques ont été développées pour garantir la sérialisabilité des transactions :
 - **Contrôle de concurrence pessimiste**: des techniques de **prévention** des conflits (En tentant de prévenir l'apparition de cycles) qui reposent sur l'idée que les conflits sont fréquents et qu'il faut les traiter le plus tôt possible.
 - => basée sur le verrouillage
 - **Contrôle de concurrence optimiste**: des techniques de **détection**
 - On suppose que les conflits sont rares et que l'on peut accepter de ré-exécuter les quelques transactions qui posent problèmes.
 - les conflits sont détectés à la confirmation de la transaction et leurs effets sont annulés.
 - => basée sur estampillage

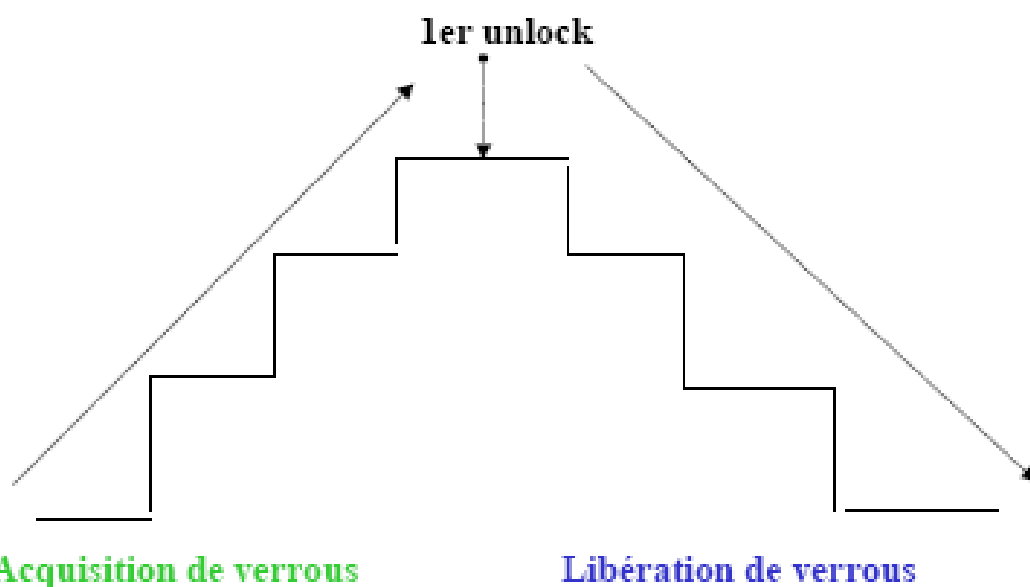
6- Techniques de contrôle de concurrence

A- Approches pessimistes : Le verrouillage

1. Protocole de verrouillage à 2 phases-V2P

Deux phases

- Phase d'acquisition de verrous (*growing phase*)
- Phase de libération de verrous (*shrinking phase*)
- **Aucun verrouillage dans une transaction après un déverrouillage**



2. Principe du verrouillage 2 phases

- ◆ Dès qu'une transaction accède à des données, elle utilise un verrou :
 - On pose des **verrous** de plusieurs types pour empêcher une utilisation potentiellement conflictuelle des ressources (nuplets)
 - Un verrou est posé jusqu'à la fin de la transaction en cours: COMMIT ou ROLLBACK = relâcher tous les verrous placés par une transaction.
- ◆ Modes de verrouillage : en lecture / écriture (les plus simples)
 - Partagé (S) : demandé avant de lire une donnée
 - Dans ce mode, une transaction peut lire un objet mais non l'écrire. **C'est un verrou en lecture.**
 - Exclusif (X) : demandé avant de modifier une donnée
 - C'est un **verrou en écriture** (une transaction peut lire et écrire l'objet.)
 - **Seulement la transaction T peut mettre à jour la donnée E et tout autre accès est interdit.**


2. Principe du verrouillage 2 phases

- ◆ Exemple:
 - 1- Begin transaction
 - 2- Update employees set salaire = salaire * 1.05
 - 3- select * from departement
 - 4- end transaction
- ➔ 2- l'utilisateur installe un **verrou exclusif** sur la table employees
 - 3- l'utilisateur installe un **verrou partagé** sur la table departement
 - 4- l'utilisateur relâche tous les verrous.

2. Principe du verrouillage 2 phases

• Compatibilité des opérations de Lectures/écritures et verrous

	L	E
L	V	F
E	F	F



Verrou demandé \ Verrou détenu	S	X
S	Accordé	Mise en attente
X	Mise en attente	Mise en attente

Exemple 2

- ♦ Séquence d'opérations reçue: H: R1 [x] R2 [y] W1 [y] c1 W2 [y] c2
 - Conflits: R2 [y] - W1 [y], W1 [y] - W2 [y]
 - forment un cycle → non sérialisable
 - Ré-écriture de l'histoire avec les verrous utilisés (Sli pour une lecture non suivie par une écriture et Xli pour une écriture:
 - ♦ H: S11[x], R1 [x], S12[y] R2 [y], X1[y] W1 [y] c1 X12[y] W2 [y] c2
 - Exécution
 - ♦ R1 [x] : acceptée, sous-verrou de lecture sur x donné à T1
 - ♦ R2 [y] : acceptée, sous-verrou de lecture sur y donné à T2
 - ♦ W1 [y] : bloquée (conflit avec R2 [y]), le sous-verrou de lecture sur y donné à T2 empêche de donner le sous-verrou d'écriture sur y à T1
 - ♦ c1 : bloquée, car le blocage de W1 [y] a bloqué toute la transaction T1
 - ♦ W2 [y] : acceptée, sous-verrou d'écriture sur y donné à T2
 - ♦ c2 : acceptée, fin de T2 et libération de tous les verrous pris par T2
- déblocage de W1 [y] et c1 qui sont acceptées
- Résultat: R1 [x] R2 [y] W2 [y] c2 W1 [y] c1
- Conflits: R2 [y] - W1 [y], W2 [y] - W1 [y]
- :pas de cycle → sérialisable

3. Problèmes du verrouillage 2 phases

- ♦ Un ordonnancement construit avec 2PL est sérialisable:
 - Garantit un graphe de précedence sans circuit
- ♦ Problèmes
 - Dégrade les performances de la base de données
 - A utiliser avec parcimonie
 - Le moins longtemps possible
 - Verrou mortel (interblocage)
 - Pour éviter l'interblocage → annulation de transactions, afin de libérer des verrous

Interblocage

- ♦ L'impasse générée par deux transactions (ou plus) qui attendent, l'une, que des verrous se libèrent, alors qu'ils sont détenus par l'autre ⇔ Attente mutuelle
- ♦ → circuit d'attente entre transactions : Détection : graphe des attentes (comme le graphe de précedence)
 - T_i attend T_j si T_i demande un verrou détenu par T_j, Si un cycle apparaît, on a un verrou mortel !

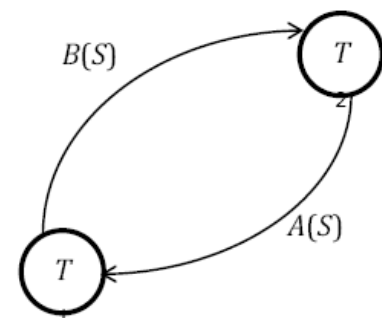
Verrouillage en écriture placée par T₁ sur A

← Verrouillage en écriture placée par T₂ sur B

Blocage de T₁
Car B est exclusivement Verrouillée par T₂

	T ₁	T ₂
	lock X A	
		lock X B
	lock S B	
	attente	lock S A
	attente	attente

Blocage de T₂
Car A est exclusivement Verrouillée par T₁



graphe d'attente

Interblocage-exemple

- ◆ Soit l'ordonnement suivant: R1 [x] W2 [y] W2 [x] W1[y] c1 c2
 - R1 [x] : T1 obtient le sous-verrou de lecture sur x
 - W2 [y] : T2 obtient le sous-verrou d'écriture sur y
 - W2 [x] : T2 bloquée par T1 , en attente du sous-verrou d'écriture sur x
 - W1 [y] : T1 bloquée par T2 , en attente du sous-verrou d'écriture sur y

T1	T2
SL(x)	
R(x)	XL(y)
	W(y)
	XL(x)
XL(y)	W[x]
R(y)	

- Interblocage ("deadlock")
- Pour éviter l'interblocage => annulation de transactions, afin de libérer

Résolution de l'interblocage

◆ Deux approches

- Prévention
 - Toutes les ressources nécessaires à la transaction sont verrouillées au départ
 - Problème : cas des transactions qui ne démarrent jamais !
- Détection :
 - On inspecte à intervalles réguliers le graphe d'attente pour détecter si un interblocage s'est produit.
 - ◆ Dans ce cas on défait l'une des transactions bloquées et on la relance un peu plus tard.
 - On annule une transaction dont le temps d'attente dépasse un certain seuil, et on la relance un peu plus tard.
 - ◆ Problème: paramétrage fin nécessaire pour la durée limite

4. Amélioration 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

5. Gestionnaire de verrouillage

- ◆ Les demandes de verrouillage et de déverrouillage sont gérées par le gestionnaire de verrouillage
 - ◆ La table des verrous pour 1 granule comporte:
 - Nombre de transactions ayant un verrou
 - Type de verrou (shared ou exclusive)
 - Pointeur vers la queue de la file des demandes de verrous
- ◆ Verrouillage et déverrouillage sont des opérations atomiques
- ◆ Notion d'upgrade d'un verrou: une transaction détenant un verrou partagé peut demander à transformer celui-ci en un verrou exclusif.

6. La granularité du verrouillage

- ♦ La granularité influe sur les performances de l'algorithme de verrouillage
- ♦ Granule plus grand (table des verrous + petite)
 - Meilleure Performance \Rightarrow un surcoût négligeable
 - Peu de concurrence \Rightarrow un parallélisme plus faible
- ♦ Granule plus fin (table des verrous importante)
 - Moins Meilleure Performance/ \Rightarrow un surcoût en verrouillage/plus de complexité
 - Plus de concurrence \Rightarrow un parallélisme élevé/ moins de risque de conflit
- ♦ Exemple:
 - Si une transaction veut modifier 95% d'une table, il est préférable de verrouiller toute la table que demander un verrou pour chaque ligne des 95%

Exemple

- **update** employé **set** salaire = salaire * 1.05
 - **update** employé **set** salaire = salaire * 1.05 **where** salaire > 10000;
- **verrou sur table lorsque:**
- la requête touche une table entière
 - la requête n'est pas très restrictive
 - la requête n'utilise pas des structures indexées

7. Conclusion verrouillage

- ♦ Approche pessimiste
 - Prévient les conflits
 - Assez coûteuse
 - Assez complexe
- ♦ Approche retenue dans tous les SGBD industriels
 - D'autres techniques existent → Voir les approches optimistes (estampillage)

Exercices

6- Techniques de contrôle de concurrence

B. Techniques optimistes: Estampillage

1. Ordonnancements par Estampillage

- ♦ Le but est d'avoir des ordonnancements sérialisables équivalents à l'ordre chronologique des transactions (unique)
- ♦ Principe d'estampillage
 - A chaque transaction est associée un numéro distinct, appelé 'estampille' (Timestamp) [ST]
 - Ce numéro est donné de manière croissante aux transactions selon leur temps de lancement (relatif au moment où elles "arrivent"),
 - (l'heure système ou bien un simple compteur)
 - $T_i \text{ avant } T_j \Leftrightarrow ST(T_i) < ST(T_j)$: T_i est la plus vieille des deux ou T_j est la plus jeune

1. Ordonnancements par Estampillage

Conservation des estampilles

- ♦ A chaque item B de la BD, on associe 2 estampilles:
 - ♦ $L_ST(B)$: Estampille de la dernière transaction ayant lu l'item B (la lectrice la plus récente) ou 0.
 - ♦ $E_ST(B)$: Estampille de la dernière transaction ayant écrit l'item B (l'écrivaine la plus récente) ou 0.

Contrôle d'ordonnement:

Si T_i désire accéder (sur un granule de concurrence):

- En écriture: Sa valeur d'estampille doit être $> \text{Writer}$ et $> \text{Reader}$
 - $ST(T_j) > E_ST(x_i)$ et $ST(T_j) > L_ST(x_i)$, $\Leftrightarrow T_j$ pourra accéder en écriture.
- En lecture: sa valeur d'estampille doit être $> \text{Writer}$:
 - $ST(T_j) > E_ST(x_i)$, $\Leftrightarrow T_j$ pourra accéder en lecture.

Exemple

Solent les transactions T_1, T_2, T_3, T_4 et T_5 avec les estampilles resp. 1, 2, 3, 4 et 5.

L'ordonnement suivant représente une situation où T_2 et T_4 sont annulées

T_1	T_2	T_3	T_4	T_5
<i>Lire(Y)</i>	<i>Lire(Y)</i>	<i>Lire(Y)</i> <i>Ecrire(Y)</i> <i>Lire(Z)</i> <i>Ecrire(Z)</i>		<i>Lire(X)</i>
<i>Lire(X)</i>	<i>Lire(Z)</i> <i>abort</i>		<i>Ecrire(Z)</i> <i>abort</i>	<i>Lire(Z)</i>
				<i>Ecrire(X)</i> <i>Ecrire(Z)</i>

2. La Certification Optimiste

♦ Les contrôles s'effectuent seulement **en fin de transaction**

- **Phase d'accès:** le CC garde les références OID (Object Identifier) des objets lus/écrits par la transaction.
 - Chaque transaction lit les valeurs des items qu'elle veut dans des copies de sa mémoire locale. Elle modifie dans les copies et non la BD
- **Phase de certification:** Le CC vérifie l'absence de conflits (L/E ou E/E même objet) avec les transactions certifiées pendant la phase d'accès. S'il y a conflit, la certification est refusée et la transaction est défaite puis reprise. (vérifier les circuits dans les G. précédentes)
- **Phase d'écriture (commit) enregistrement dans la BD** 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)

Exemple

- Séquence reçue: $l_1[x] e_2[x] l_3[x] l_2[x] e_1[x] \dots$

– Estampille = l'indice de la transaction

T1		T2		T3	
L[x]	R_ST(x)=1				

- Résultat: $l_1[x] e_2[x] l_3[x] l_2[x] e_1[x] r_1 l_4[x] e_4[x] \dots$

3. Pour résumer

- ♦ Technique efficace si les conflits sont rares
- ♦ L'estampillage permet d'éviter les blocages (les transactions sont exécutées ou bien annulées)
- ♦ Il garantit la sérialisabilité puisque tous les arcs sont de la forme $T_i \rightarrow T_j$ avec $ST(T_i) < ST(T_j)$
- ♦ Par contre, le problème de récupérabilité persiste.
 - Si T_i est annulée alors que T_j a lu une valeur écrite par T_i , alors T_j doit aussi être annulée.
 - Si T_j a déjà validé, alors l'ordonnancement n'est pas récupérable.
 - Mais si les écritures ne sont pas faites encore sur la BD (mais faites uniquement sur les copies), les annulations en cascade sont évitées
- ♦ Si on annule une longue transaction, on risque de perdre beaucoup de temps pour la relancer de nouveau
- ♦ Roll back fréquent: indication que cette technique est inappropriée pour l'environnement en question

Références

Livres:

1. G. GARDARIN, Gestion de transactions.

Supports de cours

1. T. SUDPARIS: Partie 4: ACID
2. P.RIGAUX. Les Transactions
3. N.DURAND, Cours 6 : Introduction à des notions avancées (Index, Déclencheurs, Transactions). Université de la Méditerranée - Aix-Marseille II.
4. H.OUNALLI, gestion des transactions, Faculté des Sciences de Tunis
5. M.BOUGHANEM, Chapitre 4 : Transactions et gestion de la concurrence d'accès. Institut de Recherche en Informatique de Toulouse, IRIT
6. O.PAPINI, Cours 6 : Concepts avancées : Déclencheurs, ESIL Université de la méditerranée.