Systèmes de Gestion des Bases de Données

Accès concurrents et Sécurité de fonctionnement

Nacer Boudjlida

Introduction à la notion de transaction

- Partage des ressources dans les systèmes d'exploitation
- Base de données accédée par plusieurs utilisateurs (ou programmes)
- *Sous-système de gestion des accès concurrents*
- *Techniques* : Verrouillage, exclusion mutuelle
- *Ressources partagées* : Données et méta-données
- *Transaction* $\simeq$ Programme (accès, modification)
- *Accès concurrents* $\longrightarrow$ Problèmes

Transactions : problème 1 : (Non) Atomicité

- *Virement de compte à compte* : {C1 = C1-m; C2 = C2 + m}
  1)
  2)
  3)
  4)
  5)
  6)
- Si arrêt du programme avant *Ecrire(y)* : Incohérence
- *Atomicité* : *"Tout ou rien"*
- *"Tout"* $\Longrightarrow$ *Durabilité* (Permanence des modifications)
- *"Rien"* $\Longrightarrow$ Annulation des modifications

Transactions : problème 2 : (Non) Isolation

- *Transaction T1* : Débiter X de N
- *Transaction T2* : Créditer X de M

T1	*T2*
$x \longleftarrow Lire(X)$	$y \longleftarrow Lire(X)$
$x \longleftarrow x - N$	$y \longleftarrow y + M$
$X \longleftarrow Ecrire(x)$	$X \longleftarrow Ecrire(y)$

Problème 2 : (Non) Isolation

- X = 100; N = 10; M = 50

Exécution avec entrelacement	X_{base}	x_{T1}	y_{T2}
T1 :	100		
T1 :			
T2 :			
T2 :			
T1 :			
T2 :			

- Perte de mise à jour : X = X-N
- $\Longrightarrow$ *Isolation* : pas d'interférence pendant l'exécution

Problème 3 : (Non) Cohérence

- *Contrainte* : A = B

T1 : $\{A = A + 1; B = B + 1\}$	*T2* : $\{A = A \times 2; B = B \times 2\}$
t11 : $A_{T1} \longleftarrow Lire(A)$	t21 : $A_{T2} \longleftarrow Lire(A)$
t12 : $A_{T1} \longleftarrow A_{T1} + 1$	t22 : $A_{T2} \longleftarrow A_{T2} \times 2$
t13 : $A \longleftarrow Ecrire(A_{T1})$	t23 : $A \longleftarrow Ecrire(A_{T2})$
t14 : $B_{T1} \longleftarrow Lire(B)$	t24 : $B_{T2} \longleftarrow Lire(B)$
t15 : $B_{T1} \longleftarrow B_{T1} + 1$	t25 : $B_{T2} \longleftarrow B_{T2} \times 2$
t16 : $B \longleftarrow Ecrire(B_{T1})$	t26 : $B \longleftarrow Ecrire(B_{T2})$

- <t11; t12; t13; t21; t22; t23; t14; t15; t16; t24; t25; t26> : Correct
- <t21; t22; t11; t12; t23; t13; t14; t15; t16; t24; t25; t26> : Incorrect

Problème 4 : (Non) Permanence des modifications

a) *(Non) Répétabilité des lectures*

Transaction T1	Transaction T2
$a \longleftarrow Lire(A)$	
	$b \longleftarrow Lire(A)$
	Imprimer(b)
$a \longleftarrow a + 100$	
$A \longleftarrow Ecrire(a)$	
	$b \longleftarrow Lire(A)$
	Imprimer(b)

- Impressions de T2 : valeurs différentes

Problème 4 : (Non) Permanence des modifications

b) *Tuples "fantômes"*

Transaction T1	Transaction T2
	1. $V \longleftarrow Lire(A = \{a < 4000\})$
	2. *Imprimer(V)*
3. Supprimer tout a tel que $a < 4000$	
	4. $V \longleftarrow Lire(A = \{a < 4000\})$
	5. *Imprimer(V)*

- Deuxième impression de V vide

Transactions et niveaux d'isolation : SQL 92

- *Niveau 0* : lectures "sales" permises
  - D en cours de modification par T
  - Transactions autres que T, avant validation par T :
    - bloquées en modification
    - autorisées à lire D
- *Niveau 1* : lectures "sales" interdites
- *Niveau 2* : non répétabilité des lectures avant validation
- *Niveau 3* (Par défaut) : empêcher les tuples fantômes

Transactions et reprise

- Nécessité de contrôler les accès concurrents
- *Causes d'échec des transactions* :
  - Erreur logiciel ou matériel
  - Avortement par le sous-système de gestion de la concurrence
  - "Catastrophe naturelle", etc.
- Nécessité de rétablir la cohérence : *Reprise*
  - Revenir à l'état cohérent le plus proche de l'instant de l'incident
  - Mécanisme : *Journal ou Log* ("Histoire des transactions")

Schéma d'exécution des transactions

- Transactions de mise à jour : séquences
  1. Lecture (Base $\longrightarrow$ Mémoire centrale)
  2. Traitement (en mémoire centrale)
  3. Écriture (Mémoire centrale $\longrightarrow$ Base)
- *Politique "du tout ou rien"* (**A**tomicité) :

  $Action_1$
  ...
  $Action_n$
  Si toutes les actions se sont bien passées
  Alors Valider la transaction (*COMMIT*)
  Sinon Annuler ses effets (*ROLLBACK*)
  Finsi

Schéma d'exécution des transactions

- *Validation* : rendre effectives les modifications
- *Annulation* : défaire les actions de la transaction
- *Diagramme d'états d'une transaction*

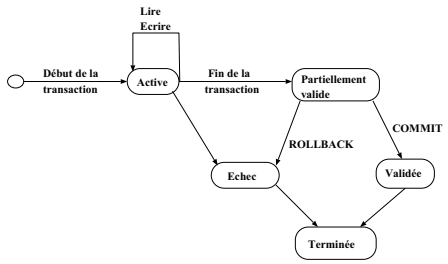

Schéma d'exécution des transactions

- *Validité partielle* :
  - Des techniques testent l'absence d'interférences
  - Des protocoles de reprise s'assurent qu'ils peuvent effectivement enregistrer les modifications
- Parfois : Annulation/Ré-exécution (UNDO/REDO) d'une opération

Journalisation, points de validation et de contrôle

- *Journal* :
  - Trace des opérations effectuées sur la base
  - Support non volatile + Archivage périodique
- *Types d'informations du journal*, pour chaque transaction :
  - <début_transaction, identification de T> ;
  - <écriture, identification de T, donnée concernée, ancienne valeur, nouvelle valeur> ;
  - <lecture, identification de T, donnée concernée>[a] ;
  - <COMMIT, identification de T>.

[a] Certains systèmes ne conservent pas de trace des opérations de lecture.

Journalisation, points de validation et de contrôle

- *Point de validation* :
  - Instant de journalisation de <COMMIT, identification de T>
  - Les actions de T ont été réussi
  - + Écritures dans le journal réussies
  - *Au-delà de ce point* : T est valide et ses MaJ sont permanentes
  - *En cas d'incident* : le système sait quoi
    - annuler ("histoire en arrière")
    - relancer, éventuellement ("histoire en avant")

Journalisation, points de validation et de contrôle

- *Points de contrôle*
  - Inscrits périodiquement dans le *Log*
  - *Période* : intervalle de temps, nombre donné de mise à jour ou combinaison des deux
  - *Prise de point de contrôle* :
    1.
    2.
    3.
    4.

  $\Longrightarrow$ les actions des transactions dont le point de validation dans le journal précède un point de contrôle pourront ne pas être ré-exécutées en cas d'incident

Points de validation et Points de contrôle

Points de validation et Points de contrôle

Transactions : Sérialisabilité et plan d'exécution

- Gestion de la concurrence fondée sur la *sérialisabilité* :

  L'exécution concurrente de n transactions doit être équivalente à (avoir le même effet pour chaque transaction que) leur exécution séquentielle

- *Propriétés (ACID)* à assurer par un gérant de transactions :
  1. *Atomicité* : politique du "tout ou rien" ;
  2. *Cohérence* : satisfaction des contraintes d'intégrité ;
  3. *Isolation* : pas d'interférence entre les transactions ;
  4. *Durabilité* : permanence des modifications effectuées.

Sérialisabilité et plan d'exécution

- A et I : par la méthode de reprise
- C : par le programmeur et le sous-système d'intégrité
- D : par les mécanismes de reprise et de contrôle de la concurrence
- *Sérialisabilité* : par le contrôle de la concurrence
- *Plan d'exécution* $\mathcal{P}$ (ou *schedule*) de n transactions :

  Ordre sur les opérations des transactions tel que pour toute transaction T_k de $\mathcal{P}$, si une opération O_i précède une opération O_j dans T_k alors O_i précède aussi O_j dans $\mathcal{P}$

- Possibilité d'entrelacement des transactions $\Longrightarrow$ $\mathcal{P}$ non unique $\Longrightarrow$ n'autoriser que les plans d'exécution corrects

Sérialisabilité et plan d'exécution

- Exécution séquentielle = *Succession* = Plan correct

  Un *plan d'exécution* $\mathcal{P}$ des transactions $T_1, \cdots, T_n$ est une *succession* (ou *serial schedule*) s'il existe une permutation Π de (1, …, n) telle que P $= < T_{\Pi(1)}; \cdots; T_{\Pi(n)} >$

- *Exécution sérialisable* : Plan correct

  Une *exécution* de $T_1, \cdots, T_n$ est *sérialisable* si et seulement si elle donne, pour chaque T_i, le même résultat qu'une succession

- *Problème* : n'autoriser que les exécutions sérialisables
  - Protocoles pessimistes : verrouillage, estampillage
  - Protocoles optimistes : contrôle à la fin d'une transaction

Contrôle de la concurrence : Verrouillage

- *Verrou* : Variable associée à un *granule* et dont la valeur indique le type d'opération possible sur un granule
- *Granule* : unité de verrouillage
  - Base, Relation, Partie de relation, etc.
- *Petite taille* :
  (+) ↗ degré concurrence
  (-) ↗ temps de son contrôle
- *Grande taille* :
  (-) ↗ temps d'attente

Contrôle de la concurrence : Types de verrous

1. *Verrou binaire* :
   - Deux états : Libre | Occupé
   - (-) Une seule transaction détient un granule
2. *Verrou partagé (Share) et verrou exclusif (Exclusive)* :
   - Accès multiples en lecture
   - En mise à jour : Un accès [+ des attentes]
   - Compatibilité des verrous :

	Partagé	*Exclusif*
Partagé		
Exclusif		

Contrôle de la concurrence : Types de verrous

3. *Verrou d'intention* :
   - Transaction demande un verrou de mise à jour ≃ verrou de lecture avec *intention* d'écriture
   - Compatibilité :

	Partagé	*Exclusif*	*Mise à jour*
Partagé	Oui	Non	Non
Exclusif	Non	Non	Non
Mise à jour	Oui	Non	Non

Gestion des transactions : Verrouillage à deux phases

- *Two Phase Locking Protocol* : le plus souvent implanté
  1. *Phase 1* (expansion) : acquisition
  2. *Phase 2* (rétrécissement) : libération et plus aucune autre acquisition
- Garantie de la sérialisabilité
- Mais verrouillage $\Longrightarrow$ Risques :
  1. Interblocage *(Deadlock)*
  2. Famine *(Livelock)*
- **Note :** *Two Phase Locking strict*

Verrouillage et Interblocage

- Attentes mutuelles
- *Prévention* : Imposer à toute transaction de verrouiller en avance tous les éléments dont elle a besoin et n'apposer aucun verrou si un de ces éléments n'est pas libre
- *Détection* : Cycle dans un graphe d'attente
  - *Nœuds* : Transactions
  - *Arc* $T1 \longrightarrow T2$: T1 *attend* un granule détenu par T2
  - Présence de cycle :
    1. *Tuer* une transaction
    2. Défaire ses actions
    3. Libérer ses ressources

Verrouillage, Interblocage et Famine

- *Exemple d'interblocage* :
- *Famine* : Attente infinie
  - *Exemple* : Priorité toujours aux mêmes transactions
  - *Solutions* : Priorités dynamiques ou *FIFO*

Contrôle de la concurrence : Estampillage

- *Estampille* : Identifiant de transaction (date de début)
- Technique fondée sur un ordre sur les *estampilles* : sérialisabilité sans interblocage
  - Plan d'exécution sérialisable s'il se conforme à l'ordre
  - *Succession* équivalente = transactions ordonnées sur les estampilles
- Granule G "marqué" par la date t de la dernière transaction qui y a accédé
- Actions sur G estampillées t', $t' \geq t$: autorisées
- Actions estampillées t'', $t'' < t$: violation de la sérialisabilité $\Longrightarrow$ Rollback

Contrôle de la concurrence : Techniques "optimistes"

- Pas de contrôle pendant l'exécution des transactions
- Mises à jour locales à la transaction
- Fin de transactions : test de non violation de la sérialisabilité
- Adaptées aux transactions ayant peu d'interférence

Reprise en cas d'incident (*Recovery*)

- Reconstruire un état cohérent à partir "du passé" (*Log*)
- État le plus proche possible de l'instant de l'incident
- Stratégie de reprise dépend de la gravité de l'incident

1. Reprise "à foid : si dégâts importants
   (a) Charger une sauvegarde de la base
   (b) Re-exécuter les transactions valides des journaux
2. Reprise "à chaud" : Défaire [et refaire] des actions

Reprise “à chaud”, Technique 1 : Mise à jour différée

- Mise à jour effective après le point de validation de la transaction
- $\simeq$ Modifications de données *“au brouillon”* (copies)
- *Cas échec* : base inchangée
- *Cas succés* : Substitution Copie/“Original” (*Shadow paging*)
- *Cas incident au moment de la substitution* : Refaire certaines actions de transactions validées

Reprise “à chaud”, Technique 1 : Mise à jour différée

Reprise “à chaud” : Technique 2 : Mise à jour immédiate

- Journalisation des modifications avant écriture dans la base avant le point de validation
- *Write Ahead Log Protocol* :
  - Implanté dans la plupart des SGBD
  - Permet la reprise même en cas d’incident entre l’écriture dans le journal et l’écriture dans la base

Write Ahead Log Protocol

WAL et reprises

- *Cas reprise à chaud* : Annuler les transactions non validées (cf. Log)
- *Cas reprise à froid* :
  1. Charger une base cohérente (exemple : B1)
  2. Charger les journaux adéquats (exemple : J2, J3)
  3. Ré-exécuter automatiquement les transactions validées
- *Remarque* : cascade de ROLLBACK
  - Rollback T1
  - T2 a utilisé des valeurs modifiées par T1

Gestion des transactions : un peu de Théorie

- Transaction (*A.C.I.D.*)
- Validation (*COMMIT*)
- Annulation (*ROLLBACK*)
- Journalisation (*LOG*)
- *Sérialisabilité* : Exécution concurrente de n transactions “équivalente” à une exécution séquentielle

Gestion des Transactions : un peu de Théorie

1. Transaction : Définitions et problèmes
2. Notion de sérialisabilité
3. Méthodes de contrôle de la concurrence

1- Transactions : Définitions

- Transaction : Séquence indivisible d'actions
- Fin d'une transaction par :
  - Succès/Validation : Effets de ses opérations sur les objets deviennent définitifs ;
  - Échec/Abandon : Annulation des effets sur les objets.
- Événements pour un système transactionnel :
  - Opérations (lire, écrire);
  - Valider/Annuler.

1- Transactions : Définitions

- Transaction $T_i = (E_i, <_i)$
- E_i = {Événements associés}
- $<_i$:
  - Ordre *partiel* sur les E_i
  - Ordre *total* sur les opérations de E_i qui concernent un même objet

1- Transactions : Définitions

- *Exécution concurrente ou histoire d'un ensemble $\mathcal{T}$ de transactions* : $T_1, \cdots, T_n = (E, <)$
  1. $E = \cup_{(1 \leq i \leq n)}(E_i)$;
  2. $<$: Ordre partiel sur E ;
  3. L'ordre $<_i$ est conservé dans $<$:
  4. Les opérations sur un même objet sont totalement ordonnées :

2- Notion de Sérialisabilité

- Sérialisabilité : Équivalence exécution concurrente, exécution séquentielle
- Définitions formelles $\longrightarrow$ Techniques et méthodes
- Plan :
  1. Exécution sérialisable
  2. Exécution sérialisable commutable
  3. Exécution sérialisable stricte
  4. Conflits et graphe de dépendances
  5. Modes de mise à jour

2.1- Exécution sérialisable

- *Exécution sérialisable* :

  Une exécution concurrente de transactions validées est sérialisable si et seulement si il existe une exécution *en série équivalente*
- *Exécution en série* (serial schedule) : Si pour tout couple (T_i, T_j) de transactions, tous les événements de l'une précèdent ceux de l'autre dans l'ordre $<$.
- *Exécutions équivalentes* : Deux exécutions d'un ensemble T de transactions sont équivalentes si et seulement si,:
  1. elles sont constituées des mêmes événements ;
  2. elles produisent le même état final des objets et les mêmes résultats pour les transactions.

2.1- Exécution sérialisable : Exemple

2.1- Exécution sérialisable : Exemple

2.2- Exécution sérialisable commutable

- Cas particulier d'exécution sérialisable
- Deux opérations O_1, O_2 sur un objet x commutent si :

  $\forall x_0$, état initial de x, $\forall T_i \ \forall T_j$

  $\{O_1^{Ti}(x); O_2^{Tj}(x)\}$ a le même effet que $\{O_2^{Tj}(x); O_1^{Ti}(x)\}$
- *Plus formellement* :
  - $Post(x, x_0, Op)$: effet de Op sur x, étant donné x_0 ;
  - $Post(T, x_0, Op)$:
    - Effet sur T de l'exécution de Op sur x_0
    - Paramètres/Valeurs retournés par Op à T

2.2- Exécution sérialisable commutable

- La *Composition* :
  -
  -
- (Op_1, Op_2) *sont commutatifs si* :
  1.
  2.
  3.

2.2- Exécution sérialisable commutable

- Notation : Commute(Op_1, Op_2)
- Remarques :
  1. Commute(lire(x), lire(x)) : toujours
  2. Commute(écrire(x_1), écrire(x_2)) si $x_1 = x_2$
- *Exécution de transactions validées commutable* si et seulement si :
  $\forall x \ Op_1^{Ti}(x) < Op_2^{Tj}(x)$
  $\longrightarrow (Valider(T_i < Op_2^{Tj}(x)) \vee Commute(Op_1, Op_2)$
- *Exemple* :

B.2.2- Exécution sérialisable commutable (fin)

Une exécution commutable est sérialisable : il existe toujours une exécution en série équivalente à l'exécution commutable (c'est l'exécution en série des transactions dans l'ordre de leurs validations).

2.3- Exécution sérialisable stricte

- Cas particulier d'exécution commutable ;
- Opérations limitées à *lire, écrire* ;
- Sans exploitation des paramètres, seules *(lire, lire)* commutent ;
- De la définition d'une exécution commutable, on a : $\forall x\ \forall T_i,\ \forall T_j$
  –
  –
  –

2.3- Exécution sérialisable stricte (fin)

- Exécution stricte caractérisée par :
  1. Dès que T_i lit un objet, d'autres transactions peuvent le lire mais pas l'écrire avant la fin de T_i ;
  2. Dès que T_i écrit un objet, aucune transaction ne peut le lire ou l'écrire tant que T_i n'est pas terminée.
- Remarque :
  – Généralisable à d'autres opérations de consultation ($\simeq$ *lire*) et de modification ($\simeq$ *écrire*) ;
  – Exploitation de la seule la commutativité des opérations de consultation.

2- Notion de Sérialisabilité : Plan

- Plan :
  1. Exécution sérialisable
  2. Exécution sérialisable commutable
  3. Exécution sérialisable stricte
  4. Conflits et graphe de dépendances
  5. Modes de mise à jour

2.4- Conflits et Graphe de dépendances

- Graphe de dépendances : Approche théorique permettant de valider les méthodes pratiques de contrôle de la concurrence
- T_i et T_j sont en *conflit* s'il existe x accédé par $Op_1^{Ti}(x)$ et $Op_2^{Tj}(x)$ et que $\neg Commute(Op_1,\ Op_2)$.
- *Graphe de dépendances* :
  $T_i \longrightarrow T_j$ (T_j dépend de T_i) si et seulement si un conflit existe entre T_i et T_j et si $Op_1^{Ti}(x) < Op_2^{Tj}(x)$
- *Sérialisabilité et Graphe de dépendances* :
  Si le graphe de dépendances ne comporte pas de circuit, alors l'exécution est sérialisable.

2.4- Conflits et Graphe de dépendances : Exemple

2.5- Modes de mise à jour et journalisation

- Annulation dépend du mode de mise à jour

1. *Mise à jour différée* :
   - Modification sur une copie des objets ("brouillon")
   - Validation par recopie du "brouillon"
2. *Mise à jour immédiate* :
   - Modification "directement" sur les objets
   - Effets peuvent être visibles avant la fin de la transaction (*dirty reads*)

2.5- Modes de mise à jour et journalisation

- cf. *Write Ahead Log Protocol*

Gestion des Transactions : un peu de Théorie

1. Transaction : Définitions et problèmes
2. Notion de sérialisabilité
3. Méthodes de contrôle de la concurrence

3- Méthodes de contrôle de la concurrence

1. *Pessismiste* : Contrôle continu
   - Détection des conflits au fur et à mesure de leur apparition
   - Si conflit : Abandon ou mise en attente de transactions
2. *Optimiste* : Contrôle par *certification*
   - À la fin de la transaction : Violation de la sérialisabilité ?
   - Adapté pour des transactions ayant peu d'interférences
   - Mal adapté avec mises à jour immédiates : des transactions peuvent utiliser des objets modifiés par une transaction qui sera annulée ($\Longrightarrow$ Risque de cascades d'abandons)

3- Méthodes pessimistes de contrôle de la concurrence

1. Estampillage : Ordre sur des dates
2. Verrouillage : Ordre par attente

3.1- Verrouillage et contrôle de la concurrence

- Verrous Exclusifs/Partagés/++ : Sérialisabilité non assurée
- cf. Types de verrous et compatibilité
- cf. *Protocole de Verrouillage à deux phases (Two-Phase Locking Protocol)* er sérailisabilité
- cf. Problèmes
  - Interblocage (*Deadlock*) : Graphe d'attente
  - Famine (*Livelock*)

3.2- Estampillage (basique) et contrôle de la concurrence

- *Estampille* : Date associée à chaque transaction
- Même estampille pour une trasnsaction et chacune de ses opérations
- Objets marqués par l'estampille de la dernière transaction qui y a accédé
- Ordre de sérialisation : Ordre total sur les estampilles
- $\simeq$ Dépendance *a priori* :
  $Estampille(T_i) < Estampille(T_j) \Longrightarrow T_i \longrightarrow T_j$.
- *Mise en œuvre* : Association à chaque objet
  1. *R(x)* : Estampille de lecture
  2. *W(x)* : Estampille d'écriture

3.2.1- Estampillage et Mise à jour immédiate

a. Conflit (écrire, lire) : Lecture de x par T estampillée t

3.2.1- Estampillage et Mise à jour immédiate

b. Conflits (lire, écrire) et (écrire, écrire) :

3.2.1- Estampillage et Mise à jour immédiate : Exemple

- T1 estampillée 10; T2 estampillée 15
- Histoire d'une exécution concurrente :
  lire$_{T1}$(x); lireT_2(y); écrire$_{T1}$(y); écrire$_{T2}$(x).
- W(x) = 8; R(x) = 12
- W(y) = R(y) = 8

3.2.1- Estampillage et Mise à jour immédiate : Exemple

3.2.2- Estampillage et Mise à jour différée

- Modification de l'estampille à la fin de la transaction
- En cas de conflit (écrire, écrire) : Validation et écriture dans l'ordre des estampilles.
- Autres méthodes : Utilisation de la sémantique (typage) des opérations.

Concurrence et Reprise : Conclusion

- Sérialisabilité et ACID"-ité"
- Verrouillage à deux phases (Interblocage)
- *Write Ahead Log Protocol*
- *Shadow Paging*
- Reprise en cas d'incident
- Sécurité par duplication (*Mirroring*)
- Organisation de l'exploitation des bases (*officier de sécurité*)
- Transactions plates : même comportement sur tous les SGBD
- Transactions imbriquées : pas de modèle d'exécution universel