Systèmes de gestion de l'information

Cours 10

Transactions et acces concurrents

Le problème

- Comme illustré dans le premier chapitre, si plusieurs programmes fonctionnent simultanément sur les mêmes données, les données peuvent devenir incohérentes.
- Par exemple, si deux applications informatiques pour réserver des billets d'avion sont exécutées à deux agences de voyage différent et les instructions de ces deux applications sont entrelacées comme ci-dessous, on se réserve deux sièges mais le nombre de sièges disponibles diminue par une seule unité:

Le problème

- Exécution entrelacée:

Moment de temps	Agence 1	Agence 2	Disp.
t1	READ A		10
t2		READ A	10
t3	A = A – 1		10
t4		A = A - 1	10
t5	WRITE A		9
t6		WRITE A	9

Le problème

- Si c'est un accès aux mêmes données, on dit que les exécutions des deux programmes sont **concurrentes** ou qu'on a un **accès concurrent**.
- Le but de ce chapitre est d'étudier les moyens d'éviter les incohérences et aussi les problèmes soulevés par les mécanismes utilisés pour cela.

Terminologie: transaction

- **Définition**: Une transaction est une séquence d'opérations effectuées sur une base de données, opérationsqui peuvent être annulées ou validées comme un ensemble.
- Dans l'exemple précédent, il y a deux transactions T1 et T2 qui sont deux exécutions différentes du même programme de réservation: T1 lancée par l'Agence 1 et T2 par l'Agence 2.

Terminologie: transaction

- Voila un autre exemple ou on a une transaction qui réserver une place et une autre qui annule une réservation de place, donc ce n'est pas nécessaire d'avoir des exécutions du même programme.
- Dans ce cas, le résultat est de nouveau une base de données incohérente car le nombre de places disponibles doit rester le même:

Transaction

◆ Réservation et annulation:

Moment de temps	Agence 1: réservation	Agence 2: annulation	Disp.
t1	READ A		10
t2		READ A	10
t3	A = A – 1		10
t4		A = A +1	10
t5	WRITE A		9
t6		WRITE A	11

Transaction

◆Sur la même base de données on peut exécuter plusieurs opérations qui travaillent avec plusieurs éléments de la base de données (pas un seul, comme dans les exemples ci-dessus).

◆Chaque transaction peut lire plusieurs données et peut écrire aussi plusieurs données de la base de données (pas nécessairement les mêmes).

Transaction

◆Les opérations d'une transaction qui ne sont ni de lecture ni d'écriture ne conduisent pas à des incohérences.

◆Par exemple l'augmentation ou la diminution de A de l'exemple précédent ne sont pas la cause des incohérences, mais l'ordre où on a fait l'écriture des résultats dans la base de données.

◆Par conséquent, dans les exemples de ce chapitre seulement les opérations de **lecture** est d'**écriture** sont figurées.

Transaction

◆ Voici une table contenant les opérations de quatre transactions concurrentes (T1-T4):

Temps	T1	T2	T3	T4
t1	READ A			
t2	READ B			
t3		READ A		
t4		READ B		
t5			WRITE B	
t6		WRITE B		
t7				READ B
t8				READ C
T9	WRITE A			
t10		WRITE C		

Remarques

Temps	T1	T2	T3	T4
t1	READ A			
t2	READ B			
t3		READ A		
t4		READ B		
t5			WRITE B	
t6		WRITE B		
t7				READ B
t8				READ C
T9	WRITE A			
t10		WRITE C		

◆Notez que nous avons:

- Des transactions qui écrivent des données qui ont été précédemment lues par elles (comme T1 qui écrit A, déjà lu)
- Des transactions qui écrivent aussi des données qui n'ont pas été précédemment lues par elles - éventuellement calculés (comme T2 qui écrit B mais aussi C)
- Des transactions qui exécutent seulement des lectures (comme T4)
- Des transactions qui exécutent seulement des écritures (comme T3)

Terminologie: Article

◆ **Définition**: Un article est une partie de la base de données qui peut être lu, écrit, verrouillé ou déverrouillé par une seule opération de type READ, WRITE, LOCK ou UNLOCK.

◆ Dans l'exemple précédent on a figure des articles symboliques comme A, B, C, etc. Dans les cas réels un article peut être:

- ❖ Une table entière
- ❖ Une ligne (ou un ensemble de lignes) d'une table
- ❖ Une cellule d'une table de la BD
- ❖ Toute autre partie de la base de données qui répond à la définition, en considérant les facilités prévues par les SGBD respectifs.

Terminologie: Article

- Exemple: le système Oracle verrouille automatiquement une ligne mise à jour par une requête UPDATE jusqu'au moment ou la transaction qui contienne la commande est validée ou annulée.
- Donc dans ce cas les articles sont des lignes d'une table.

Terminologie: Ordonnancement

- **Définition**: Un **ordonnancement** (anglais: schedule) est l'ordre dans lequel sont exécutés par le SGBD les requêtes d'un ensemble de transactions.
- Dans ce chapitre, nous figurons seulement les requêtes qui représentent une interaction avec les données de la base de données:
  - READ - lire un article
  - WRITE - écrire un article
  - LOCK (dans ses différentes formes) - verrouillage d'un article
  - UNLOCK - déverrouillage d'un article

Ordonnancement

- Pour les cas où l'opération d'écriture ne devienne pas permanente dans la base de données jusqu'à la validation, on a aussi:
  - COMMIT - valider les changements effectués par une transaction
  - ROLLBACK - annuler les changements effectués par une transaction
- Comme mentionné, les autres opérations ne doivent pas être figurées parce qu'elles n'ont pas aucune importance pour le résultat d'une exécution concurrente.

Ordonnancement

- On peut représenter un ordonnancement:
1. Sous forme d'un tableau comme dans les exemples précédents. La colonne "Temps" peut être absent, l'ordre d'exécution est de haut en bas.
2. Sous forme d'une liste.

- Pour la deuxième forme, on peut utiliser la notation suivante:
  - $R_i(A)$ - signifie: transaction T_i lit l'article A
  - $W_i(A)$ - signifie: transaction T_i écrit l'article A
- Dans ce cas l'ordonnancement (pour T1, T2 et T3) peut être écrite comme:

$R_1(A); R_1(B); R_2(A); R_2(B); W_3(B); W_2(B); R_4(B); R_4(C); W_1(A); W_1(C)$

Ordonnancement en série

- **Définition**: Un **ordonnancement** dans lequel les requêtes de chaque transaction sont successives, sans être entrelacés avec les requêtes d'autres transactions, est appelé un **ordonnancement en série** (anglais: serial schedule).
- Exemple d'ordonnancement en série pour les deux transactions T1 et T2:

$R_1(A); R_1(B); R_1(C); W_1(B); R_2(B); R_2(C); W_2(A); W_2(C)$

Requêtes de T1 Requêtes de T2

Ordonnancement en série

- Les ordonnancement en série ne produisent des incohérences dans la base de données, donc sont des 'bons' ordonnancements - en termes d'exécution concurrente.
- Par conséquent, un objectif de ce chapitre est de trouver des ordonnancements qui se comportent comme un ordonnancement en série.

Terminologie: Verrouillage

- Comme mentionné dans le premier chapitre, une méthode d'obtenir des ordonnancements qui préservent la cohérence des données est de verrouiller les articles avant leur utilisation.
- **Définition**: le verrouillage d'un article par une transaction signifie que le SGBD donne à la transaction certains droits spéciaux d'accès qui empêchent les autres transactions d'effectuer certaines opérations sur l'article.

Verrouillage

- Il y a deux types de verrous:

1. **Verrous exclusifs**: les autres transactions ne peuvent exécuter aucun type d'opération sur l'article verrouillé. Ces verrous sont appelés dans la littérature des verrous exclusifs ou verrous d'écriture (anglais: exclusive locks ou write locks).
2. **Verrous partagés**: les autres transactions peuvent exécuter certaines opérations sur l'article verrouillé. Ces verrous sont appelés dans la littérature des verrous partagés ou verrous de lecture (anglais: shared locks ou read locks).

Verrouillage: Example

- Pour écrire un article, une transaction doit avant obtenir un verrou d'écriture sur lui: aucune autre transaction ne peut pas lire ou écrire l'article jusqu'à ce qu'il soit déverrouillé.
- Pour lire un article, une transaction doit avant obtenir un verrou de lecture sur lui.
- Plusieurs transactions peuvent verrouiller simultanément le même article pour lecture, mais aucune transaction ne peut l'écrire.

Verrouillage: Example

- Une transaction peut obtenir un verrou d'écriture sur l'article seulement après que toutes les autres transactions ont déverrouillé leurs verrous partages.
- Les verrous peuvent être déverrouillés soit un par un, soit, en cas de validation (COMMIT) ou annulation (ROLLBANK), tous les verrous sont déverrouillés: ça dépend du modèle de verrouillage utilisé.

Gestion des transactions: ACID

- En termes gestion des transactions, un SGBD doit fournir certaines propriétés qui sont regroupées sous l'acronyme ACID. Cela signifie:
  - A - Atomicité
  - C - Cohérence
  - I - Isolation
  - D - Durabilité.

Atomicité

- Une transaction doit être atomique dans le sens où toutes les modifications apportées dans la base de données sont toutes validées ou toutes annulées.
- Exemple: Considérons une séquence de requêtes SQL contenant des insertions, mises à jour et suppressions des lignes de deux tables, ETUDIANT et FACULTE:

```
INSERT INTO ETUDIANT ...
UPDATE FACULTE ...
DELETE FROM ETUDIANT …
```

Atomicité

- Ces changements doivent être soit tous validés (COMMIT) soit tous annulés (ROLLBACK). Si, par exemple, seulement INSERT est validé et UPDATE et DELETE non est une violation de cette règle.
- Le système de gestion est celui qui rend disponibles les mécanismes pour assurer l'atomicité des transactions, y compris dans le cas d'incident matériel et/ou logiciel qui peut survenir lors de l'exécution d'une transaction.

Cohérence

- Une transaction qui commence à travailler sur une base de données cohérente doit au final laisser la base de données dans un état aussi cohérent.
- À cet égard, une transaction ne peut pas violer les restrictions existantes au niveau de la base de données.
- Dans la plupart des cas, ces restrictions sont modélisées comme des contraintes d'intégrité (NOT NULL, PRIMARY KEY, UNIQUE, FOREIGN KEY ou CHECK).

Cohérence

- Si une transaction contient une opération qui viole une contrainte d'intégrité, les modifications apportées par la transaction seront annulées.
- Les mécanismes pour assurer la cohérence doivent être assurés par le SGBD.

Isolation

- Une transaction doit se comporter comme si les opérations effectuées sont isolées, indépendantes des toutes opérations effectuée par une autre transaction.
- Pour cela, aucune autre transaction ne doit pas lire des données intermédiaires écrites par la transaction.

Isolation

- Par exemple, si une transaction contient la séquence suivante:

```
READ (A)      -- lect.ancienne val.de A: 5
A = A + 1
ECRITURE (A) -- écr.nouvelle val.de A: 6
READ (B)     -- lect.ancienne val.de B: 10
B = B - 1
ECRITURE (B) -- écr.nouvelle val.de B: 9
```

- Aucune autre transaction ne doit lire pour A et B deux valeur dont l'une est mise à jour et l'autre non (6 pour A et 10 pour B).

Isolation

- Les incohérences précédentes (agence de voyage) étaient dues à la violation de l'isolation. Les mécanismes pour assurer l'isolation doivent aussi être assurés par le SGBD.
-
- L'isolation est l'objet du contrôle de l'accès concurrent présenté dans les paragraphes suivants.

Durabilité

- Une fois validés avec succès, les modifications apportées par une transaction vont persister et ne peuvent pas être annulés.
- Aussi, dans le cas d'un incident matériel et/ou logiciel, les effets des transactions validées sont présents dans la base de données obtenue par la récupération après l'incident.
- De ce point de vue chaque système de gestion doit avoir les mécanismes par lesquels les effets de toutes les transactions validées sont enregistrés dans les journaux du système pour être restauré en cas d'incident.

Sérialisabilité

- On a dit déjà qu'un ordonnancement en série ne conduit pas à des incohérences.
- Dans la pratique, cependant, dans un SGBD, l'exécution concurrente de plusieurs transactions est faite avec des requêtes appartenant à divers transactions entrelacées.
- Le résultat sera correcte si l'effet est le même que celui d'un ordonnancement en série du même ensemble de transactions.
- Un tel ordonnancement est appelé **ordonnancement sérialisable**.

Ordonnancement sérialisable

- Définition: Un ordonnancement est **sérialisable** s'il a les mêmes effets dans la base de données comme un des ordonnancements en série possibles.
- Exemple 1: Un ordonnancement sérialisable et l'ordonnancement en série équivalent:

T1	T2	T3
READ A		
READ B		
		READ C
		READ D
WRITE A		
		WRITE C
	READ A	
	WRITE A	

T1	T2	T3
		READ C
		READ D
		WRITE C
READ A		
READ B		
WRITE A		
	READ A	
	WRITE A	

Ordonnancement sérialisable

- Exemple 2: Ordonnancement non-sérialisable et les deux ordonnancement en série équivalents: prenons l'exemple précédent de deux transactions qui réserve et annule une réservation:

T1	T2	A de BD
READ A		10
	READ A	10
A = A + 1		10
	A = A - 1	10
WRITE A		11
	WRITE A	9

Ordonnancement sérialisable

- Il y a seulement deux ordonnancement en série différents pour les T1 et T2 et aucun ne produisent le même résultat:

T1	T2	A de BD
READ A		10
A = A + 1		10
WRITE A		11
	READ A	11
	A = A - 1	10
	WRITE A	10

T1	T2	A de BD
	READ A	10
	A = A +1	10
	WRITE A	11
READ A		11
A = A – 1		11
WRITE A		10

Contrôle de concurrence

- **Le contrôle de concurrence pessimiste**: dans ce cas on utilise le **verrouillage** pour empêcher les transactions de modifier le même article dans le même temps.
- **Le contrôle de concurrence optimiste**: les transactions ne verrouillent pas les articles mais le SGBD vérifie pour chaque opération si les transactions sont ou non en **conflit**. Si un conflit est détecté, une des transactions est **annulée** (pour annuler le conflit) et puis redémarrée.

Sérialisabilité par verrouillage

- Pour assurer la sérialisabilité des transactions, les systèmes de gestion mettent à disposition la possibilité de verrouiller les articles.
- Si une transaction verrouille un article, les autres transactions demandant un accès à cet élément peuvent être mises en attente jusqu'à son déverrouillage.
- Il existe plusieurs modèles de verrouillage, le plus simple étant le modèle ci-dessous.

Le modèle avec LOCK et UNLOCK

- Dans ce modèle, il ya une seule commande de verrouillage, LOCK, résultant un accès exclusif à l'article par la transaction qui a exécuté la commande (autres transactions ne peuvent pas ni lire ni écrire l'article).
- Le déverrouillage se fait avec UNLOCK.
- Nous supposons en outre que, pour une transaction:
  - Elle ne verrouille un article déjà verrouillé par elle
  - Elle ne déverrouiller pas un article n'a pas été verrouillée par elle.
- Dans ce cas le **test de sérialisabilité** pour un ordonnancement S peut être effectué comme suit:

Test de sérialisabilité

- On construit le **graph de précédence G** ou:

1. Les nœuds sont les transactions
2. Si, pour un article A l'ordonnancement S contient la séquence:
   Ti: UNLOCK A
   Tj: LOCK A
   Alors on ajoute dans le graph G un arc entre le nœud Ti et le nœud Tj.
3. Si le graph **a des cycles**, alors S **n'est pas sérialisable.**
4. Si le graph **n'a pas des cycles**, S est **sérialisable** et on obtient l'ordonnancement en série équivalent est par tri topologique du graphe G

Tri topologique

- Le tri topologique pour obtenir l'ordonnancement en série équivalent O est fait dans la manière suivante:

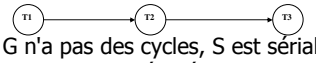
1. O = ∅ /* O est un ensemble ordonné */
2. On choisit un nœud N qui n'a pas des arcs entrants (on a des nœuds comme ça si le graph n'a pas des cycles)
3. O = O + {N}
4. N est effacé de G, aussi tous les arcs adjacents
5. GOTO 2

Exemple

- Exemple: Soit l'ordonnancement S:

T1	T2	T3
	LOCK A	
	UNLOCK A	
		LOCK A
		UNLOCK A
LOCK B		
UNLOCK B		
	LOCK B	
	UNLOCK B	

- Le graph G est:



- Comme G n'a pas des cycles, S est sérialisable et l'ordonnancement en série équivalent est:

T1; T2; T3

Verrouillage en deux phases (2PL)

- **Le protocole de verrouillage en deux phases**
- Définition: Une transaction vérifie le protocole de verrouillage en deux phases - 2PL (anglais: two phase locking) si:

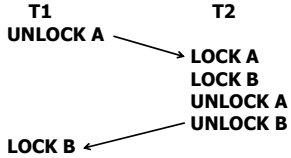
1. Avant de lire ou écrire un article, la transaction doit le verrouiller (pour lecture ou écriture).
2. A sa fin, la transaction déverrouille tous les articles qui ont été verrouilles
3. **Toutes les opérations de verrouillage précédent toutes les opérations de déverrouillage**

Verrouillage en deux phases (2PL)

- Ce protocole garantit la sérialisabilité: si toutes les transactions d'un ordonnancement S répondant aux exigences du 2PL, alors l'ordonnancement est sérialisable.
- En outre, il peut être démontré que si une transaction répondant aux exigences du 2PL on peut avoir des exécutions non-sérialisable en conjonction avec d'autres transactions:

Verrouillage en deux phases (2PL)

- Exemple: pour une transaction qui contient la séquence (qui ne vérifie pas 2PL):
  UNLOCK A
  LOCK B
- On peut avoir l'ordonnancement suivant qui n'est pas sérialisable parce que son graphe de précédence contient un cycle:



Verrouillage en deux phases (2PL)

- Le problème est que le protocole de verrouillage en deux phases détermine parfois des opérations d'annulation (ROLLBACK) en cascade.
- Dans l'exemple suivant, en cas d'annulation de T1 est nécessaire aussi une annulation de T2 parce que T2 a lu des données écrites par T1, données qui peuvent être perdues par l'annulation de T1.
- Cet ordonnancement est appelé **ordonnancement avec annulation en cascade** (Anglais: cascading aborts).

Verrouillage en deux phases (2PL)

- Ordonnancement avec annulation en cascade

T1	T2
LOCK A LOCK B	
READ A WRITE A	
UNLOCK A	
	LOCK A READ A WRITE A UNLOCK A
READ B WRITE B	
ROLLBACK	

2PL strict

- Pour éviter de tels cas, on a le **protocole de verrouillage *strict* en deux phases** comportant le déverrouillage de tous les articles bloqués **uniquement à la fin de la transaction** (pas comme étape intermédiaire).
- Dans ce cas, la transaction T2 de l'exemple précédent commence seulement après l'achèvement de la transaction T1 (T1 fait UNLOCK A seulement au final).

Concurrence optimiste

- **Le contrôle de concurrence optimiste**: les transactions ne verrouillent pas les articles mais le SGBD vérifie pour chaque opération si les transactions sont ou non en conflit.
- Si un conflit est détecté, une des transactions est annulée (pour annuler le conflit) et puis redémarrée.
- Un tel mécanisme peut être obtenu en utilisant **l'estampillage** (avec des estampilles de temps - Anglais: **timestamps**).

Estampillage

- Dans ce cas:

1. Chaque transaction a une **estampille de temps** associée (par exemple le moment de son lancement, donné par l'horloge du système)
2. Chaque article a une **estampille de lecture** et une **estampille d'écriture**, données par les estampilles des transactions qui ont fait la plus récente lecture et écriture de l'article.

- Donc, au moment ou une transaction lit ou écrit un article, son estampille de lecture ou d'écriture est mise à jour et prend la valeur de l'estampille de temps de la transaction.

Estampillage

- Le système détecte un conflit si:

1. Une transaction tente de lire un article écrit dans le futur (l'estampille d'écriture de l'article est plus récente que l'estampille de la transaction)
2. Une transaction tente d'écrire un article lu dans le futur (l'estampille de lecture de l'article est plus récente que l'estampille de la transaction)

Estampillage

- Remarque: les opérations suivantes sont valides et ne représentent un conflit:

1. Une transaction tente de lire un article lu dans le futur
2. Une opération tente d'écrire un article écrit dans le futur (dans ce cas elle n'écrit pas l'article, mais continue son exécution).

Fin