Cours de Base de Données
Cours n.11
Gestion des transactions : contrôle de concurrence

L2 Informatique - Portail Siences et Technologies

Elisabetta De Maria - http://www.i3s.unice.fr/∼edemaria/

DS4H et Laboratoire I3S, CNRS

2023-2024
Université Côte d'Azur

Transaction

Definition (Transaction)

Une action ou une suite d'actions demandée par un seul utilisateur ou programme d'application, qui lit ou met à jour le contenu de la base de données.

Une transaction est une unité logique de travail sur la base de données.

Example (Tables Personnel et ProprietéAlouer)

Personnel(numPersonnel, prénom, nom, fonction, salaire)
ProprietéALouer(numProprieté, type, pièces, numPropriétaire, numPersonnel)

Examples de transactions

```
lire(numPersonnel = x, salaire)
nouv_salaire = salaire * 1.1
écrire(numPersonnel = x, nouv_salaire)
```

a)

```
supprimer(numPersonnel = x)
pour tous les enregistrements de PropriétéALouer, numProp
début
    lire(numPropriété = numProp, numPersonnel)
    si (numPersonnel = x) alors
        numPersonnel = nouvNumPers
        écrire(numPropriété = numProp, numPersonnel)
    fin
fin
```

b)

UNIVER
CÔTE D'A

Résultats possibles d'une transaction

- Si une transaction s'achève avec succès, la transation est dite **validée** ou **confirmée** (*committed*) et la base de données atteint un nouvel état cohérent.
- Si une transaction ne s'exécute pas avec succès, elle est **avortée** (*aborted*). Dans ce cas, la base de données doit retrouver l'état cohérent qu'elle avait avant le début de la transaction. Une transaction qui subit cette réinitialisation est dite **annulée** (*rolled back*).

Propriétés de transactions

ACID (Haerder et Reuter, 1983)

- *Atomicité.* Une transaction forme une unité indivisible, soit exécutéee dans sa totalité, soit annullée dans sa totalité.
- *Cohérence.* Une transaction transforme la base de données d'un état cohérent en un autre état cohérent.
- *Isolation.* Les transactions s'exécutent de manière indépendante les unes des autres.
- *Durabilité.* Les effets d'une transaction complètement achevée (validée par un COMMIT) sont inscrits de manière durable dans la base de données et ne peuvent subir de perte à la suite d'une défaillance subséquente.

Necessité du contrôle de concurrence

Un des principaux objectifs du développement d'une base de données est de permettre à plusieurs utilisateurs d'accéder simultanément à des données partagées.

Definition (Contrôle de concurrence)

Le procédé qui consiste à gérer des opérations simultanées sur une base de données, tout en interdisant les interférences entre ces opérations.

Trois exemples de problèmes provoqués par la concurrence :

- le problème de la mise à jour perdue
- le problème de la dépendance non validée
- le problème de l'analyse incohérente

Le problème de la mise à jour perdue

Une opération de modification apparemment achevée avec succès par un utilisateur est écrasée par un autre utilisateur.

Solde de 90 € au lieu de 190 €.

Temps	T_1	T_2	$solde_x$
t_1		début_transaction	100
t_2	début_transaction	lire($solde_x$)	100
t_3	lire($solde_x$)	$solde_x = solde_x + 100$	100
t_4	$solde_x = solde_x - 10$	écrire($solde_x$)	200
t_5	écrire($solde_x$)	validation	90
t_6	validation		90

UNIVER
CÔTE D'A

Le problème de la dépendance non validée (dirty read)

Une transaction est autorisée à voir immédiatement les résultats d'une autre transaction, avant que celle-ci n'ait été validée.

Solde de 190 € au lieu de 90 €.

Temps	T_3	T_4	$solde_x$
t_1		début_transaction	100
t_2		lire($solde_x$)	100
t_3		$solde_x = solde_x + 100$	100
t_4	début_transaction	écrire($solde_x$)	200
t_5	lire($solde_x$)	⋮	200
t_6	$solde_x = solde_x - 10$	annulation	100
t_7	écrire($solde_x$)		190
t_8	validation		190

UNIVER
CÔTE D'A

Le problème de l'analyse incohérente

Une transaction lit plusieurs valeurs de la base de données mais une autre transaction en modifie certaines pendant l'execution de la première transaction.

Somme de 185 € au lieu de 175 €.

Temps	T_5	T_6	$solde_x$	$solde_y$	$solde_z$	somme
t_1		début_transaction	100	50	25	
t_2	début_transaction	somme = 0	100	50	25	0
t_3	lire($solde_x$)	lire($solde_x$)	100	50	25	0
t_4	$solde_x = solde_x - 10$	somme = somme + $solde_x$	100	50	25	100
t_5	écrire($solde_x$)	lire($solde_y$)	90	50	25	100
t_6	lire($solde_z$)	somme = somme + $solde_y$	90	50	25	150
t_7	$solde_z = solde_z + 10$		90	50	25	150
t_8	écrire($solde_z$)		90	50	35	150
t_9	validation	lire($solde_z$)	90	50	35	150
t_{10}		somme = somme + $solde_z$	90	50	35	185
t_{11}		validation	90	50	35	185

UNIVER
CÔTE D'A

Capacité de sérialisation (1)

La **capacité de sérialisation** (*serializability*) représente une aide à l'identification des transactions dont l'execution est garantie pour assurer la coherence.

Definition (Planification (schedule))

Une séquence d'opérations d'un ensemble de n transactions $T_1, T_2, \ldots, T_n$ qui preserve l'ordre des opérations dans chacune des transactions.

Definition (Planification sérielle)

Une planification où les operations de chaque transaction sont exécutées de manière consécutive, sans aucune opération interfoliée d'autres transactions.

Capacité de sérialisation (2)

Definition (Planification non sérielle)

Une planification où les opérations d'un ensemble de transactions sont exécutées de manière interfoliée.

Definition (Planification sérialisable)

Une planification non sérielle qui produit les mêmes résultats qu'une exécution sérielle.

- Si deux transactions ne font que lire des données, elles n'entrent pas en conflit et leur ordre est sans importance.
- Si deux transactions soit lisent, soit écrivent complètement des données différentes, leur ordre est sans importance.
- Si une transaction écrit dans des données et si une autre transaction lit où écrit dans ces mêmes données, alors l'ordre de leur exécution importe.

Example de planifications équivalents

Ce type de capacité de sérialisation est connu sous le terme de **capacité de sérialisation des conflits**.

Temps	T_7	T_8
t_1	début_transaction	
t_2	lire(solde$_x$)	
t_3	écrire(solde$_x$)	
t_4		début_transaction
t_5		lire(solde$_x$)
t_6		écrire(solde$_x$)
t_7	lire(solde$_y$)	
t_8	écrire(solde$_y$)	
t_9	validation	
t_{10}		lire(solde$_y$)
t_{11}		écrire(solde$_y$)
t_{12}		validation

a)

	T_7	T_8
	début_transaction	
	lire(solde$_x$)	
	écrire(solde$_x$)	
		début_transaction
		lire(solde$_x$)
	lire(solde$_y$)	
		écrire(solde$_x$)
	écrire(solde$_y$)	
	validation	
		lire(solde$_y$)
		écrire(solde$_y$)
		validation

b)

	T_7	T_8
	début_transaction	
	lire(solde$_x$)	
	écrire(solde$_x$)	
	lire(solde$_y$)	
	écrire(solde$_y$)	
	validation	
		début_transaction
		lire(solde$_x$)
		écrire(solde$_x$)
		lire(solde$_y$)
		écrire(solde$_y$)
		validation

c)

UNIVER
CÔTE D'A

Test de conflits de la capacité de sérialisation

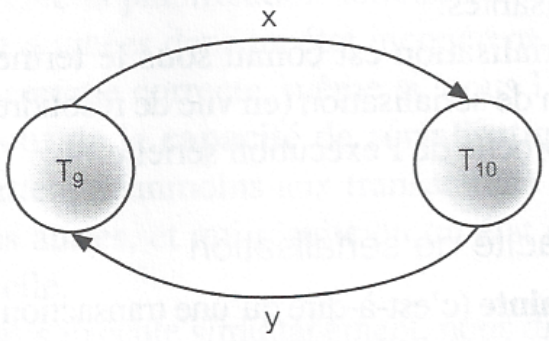
Pour une planification P, un **graphe de précédence** est un graphe dirigé G=(N,F) construit comme suit :

- Créer un nœud pour chaque transaction.
- Créer une flèche dirigée $T_i \rightarrow T_j$, si T_j lit la valeur d'un élément écrit par T_i.
- Créer une flèche dirigée $T_i \rightarrow T_j$, si T_j écrit une valeur dans un élément après qu'il a été lu par T_i.
- Créer une flèche dirigée $T_i \rightarrow T_j$, si T_j écrit une valeur dans un élément après qu'il a été écrit par T_i.

Si une flèche $T_i \rightarrow T_j$ existe dans le graphe de précédence pour P, alors dans toute planification sérielle S équivalente à P, T_i doit apparaître avant T_j. Si le graphe de précédence contient un cycle, la planification n'est pas sérialisable en vue de résoudre les conflits.

Planification sans sérialisation des conflits

Temps	T_9	T_{10}
t_1	début_transaction	
t_2	lire($solde_x$)	
t_3	$solde_x = solde_x + 100$	
t_4	écrire($solde_x$)	début_transaction
t_5		lire($solde_x$)
t_6		$solde_x = solde_x * 1.1$
t_7		écrire($solde_x$)
t_8		lire($solde_y$)
t_9		$solde_y = solde_y * 1.1$
t_{10}		écrire($solde_y$)
t_{11}	lire($solde_y$)	validation
t_{12}	$solde_y = solde_y - 100$	
t_{13}	écrire($solde_y$)	
t_{14}	validation	



UNIVERSITÉ CÔTE D'AZUR

Techniques de contrôle de concurrence

Il existe différentes techniques de contrôle de concurrence qui permettent d'exécuter des transactions en parallèle en toute sécurité, à condition de faire appel à certaines contraintes :

- les méthodes de verrouillage
- les méthodes d'estampillage
- les méthodes optimistes

Méthodes de verrouillage

Definition (Verrouillage)

Une procédure employée pour contrôler les accès concurrents aux données. Lorsqu'une transaction accède à une base de données, un verrou est susceptible de bloquer l'accès à d'autres transactions pour éviter de faux résultats.

Definition (Verrou partagé)

Si une transaction dispose d'un verrou partagé sur une donnée, elle peut lire la donnée mais pas la modifier.

Definition (Verrou exclusif)

Si une transaction dispose d'un verrou exclusif sur une donnée, elle peut lire et modifier cette donnée.

Usage des verrous

- Toute transaction devant accéder à une donnée verrouille d'abord la donnée, demandant soit un verrouillage partagé soit un verrouillage exclusif.
- Si la donnée n'est pas déjà verrouillée par une autre transaction, le verrou est accordé.
- Si la donnée est déjà verrouillée au moment de la demande, le SGBD détermine si la demande est compatible avec le verrou actuel. Si c'est un verrou partagé que la transaction demande, alors qu'un verrou partagé est déjà placé sur la donnée, la requête peut être satisfaite et le verrou est accordé ; dans le cas contraire, la transaction demanderesse doit attendre que le verrou se libère.
- Une transaction qui détient un verrou le conserve tant qu'elle ne le libère pas explicitement pendant l'execution ou implicitement lorsqu'elle se termine (par une annullation ou une validation). Ce n'est que lorsqu'un verrou exclusif est liberé que les effets de l'opération d'écriture qui a motivé le verrou deviennent visibles aux autres transactions.

Planification incorrecte du verrouillage

Reprenons les deux transations du transparent 14.
$P = \{$verrou-écriture$(T_9, solde_x)$, lecture$(T_9, solde_x)$, écriture$(T_9, solde_x)$, déverrouillage$(T_9, solde_x)$, verrou-écriture$(T_{10}, solde_x)$, lecture$(T_{10}, solde_x)$, écriture$(T_{10}, solde_x)$, déverrouillage$(T_{10}, solde_x)$, verrou-écriture$(T_{10}, solde_y)$, lecture$(T_{10}, solde_y)$, écriture$(T_{10}, solde_y)$, déverrouillage$(T_{10}, solde_y)$, validation(T_{10}), verrou-écriture$(T_9, solde_y)$, lecture$(T_9, solde_y)$, écriture$(T_9, solde_y)$, déverrouillage$(T_9, solde_y)$, validation$(T_9)\}$
Préalablement à l'execution, $solde_x = 100$ et $solde_y = 400$.
Si T_9 s'exécute d'abord : $solde_x = 220$ et $solde_y = 330$.
Si T_{10} s'exécute d'abord : $solde_x = 210$ et $solde_y = 340$.
Résultat de P : $solde_x = 220$ et $solde_y = 340$.

UNIVERSITÉ CÔTE D'AZUR

Verrouillage en deux phases

Definition (Verrouillage en deux phases (V2P))

Une transaction suit le protocole de verrouillage en deux phases si toutes les opérations de verrouillage précèdent la première opération de déverrouillage dans la transaction.

Toute transaction est divisible en deux phases : une première phase de **croissance** où elle acquiert tous les verrous mais ne peut en libérer aucun, et une phase de **résorption**, au cours de laquelle elle libère tous les verrous et ne peut plus en obtenir aucun.

- Une transaction doit acquérir un verrou sur un élément avant de faire quoi que ce soit sur celui-ci.
- Une fois que la transaction libère un verrou, elle ne peut plus demander d'autres verrous.

Éviter le problème de la mise à jour perdue à l'aide de la V2P

Temps	T_1	T_2	$solde_x$
t_1		début_transaction	100
t_2	début_transaction	verrou_écriture($solde_x$)	100
t_3	verrou_écriture($solde_x$)	lire($solde_x$)	100
t_4	ATTENTE	$solde_x = solde_x + 100$	100
t_5	ATTENTE	écrire($solde_x$)	200
t_6	ATTENTE	validation/déverrouillage($solde_x$)	200
t_7	lire($solde_x$)		200
t_8	$solde_x = solde_x - 10$		200
t_9	écrire($solde_x$)		190
t_{10}	validation/déverrouillage($solde_x$)		190

Éviter le problème de la dépendance non validée à l'aide de la V2P

Temps	T_3	T_4	$solde_x$
		début_transaction	100
t_1		verrou_écriture($solde_x$)	100
t_2		lire($solde_x$)	100
t_3		$solde_x = solde_x + 100$	200
t_4	Début_transaction	écrire($solde_x$)	200
t_5	verrou_écriture($solde_x$)	annulation /déverrouillage($solde_x$)	100
t_6	ATTENTE		100
t_7	lire($solde_x$)		100
t_8	$solde_x = solde_x - 10$		90
t_9	écrire($solde_x$)		90
t_{10}	validation/déverrouillage($solde_x$)		

Éviter le problème de l'analyse incohérente à l'aide de la V2P

Temps	T_5	T_6	$solde_x$	$solde_y$	$solde_z$	somme
t_1		début_transaction	100	50	25	
t_2	début_transaction	somme = 0	100	50	25	0
t_3	verrou_écriture($solde_x$)		100	50	25	0
t_4	lire($solde_x$)	verrou_lecture($solde_x$)	100	50	25	0
t_5	$solde_x = solde_x - 10$	ATTENTE	100	50	25	0
t_6	écrire($solde_x$)	ATTENTE	90	50	25	0
t_7	verrou_écriture($solde_z$)	ATTENTE	90	50	25	0
t_8	lire($solde_z$)	ATTENTE	90	50	25	0
t_9	$solde_z = solde_z + 10$	ATTENTE	90	50	25	0
t_{10}	écrire($solde_z$)	ATTENTE	90	50	35	0
t_{11}	validation/déverrouillage($solde_x$, $solde_z$)	ATTENTE	90	50	35	0
t_{12}		lire($solde_x$)	90	50	35	0
t_{13}		somme = somme + $solde_x$	90	50	35	90
t_{14}		verrou_lecture($solde_y$)	90	50	35	90
t_{15}		lire($solde_y$)	90	50	35	90
t_{16}		somme = somme + $solde_y$	90	50	35	140
t_{17}		verrou_lecture($solde_z$)	90	50	35	140
t_{18}		lire($solde_z$)	90	50	35	140
t_{19}		somme = somme + $solde_z$	90	50	35	175
t_{20}		validation/déverrouillage($solde_x$, $solde_y$, $solde_z$)	90	50	35	175

UNIVER
CÔTE D'A

Blocage indéfini (deadlock)

Definition (Blocage indéfini (deadlock))

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

Temps	T_{17}	T_{18}
t_1	début_transaction	
t_2	verrou_écriture($solde_x$)	début_transaction
t_3	lire($solde_x$)	verrou_écriture($solde_y$)
t_4	$solde_x = solde_x - 10$	lire($solde_y$)
t_5	écrire($solde_x$)	$solde_y = solde_y + 100$
t_6	verrou_écriture($solde_y$)	écrire($solde_y$)
t_7	ATTENTE	verrou_écriture($solde_x$)
t_8	ATTENTE	ATTENTE
t_9	ATTENTE	ATTENTE
t_{10}	⋮	ATTENTE
t_{11}	⋮	⋮

UNIVER
CÔTE D'A

Techniques permettent de gérer les verrous indéfinis

Trois techniques permettent de gérer les verrous indéfinis :

- imposer un delai imparti
- prévenir les verrous indéfinis
- détecter les verrous et les récuperer

Détection des verrous indéfinis

Le **graphe de attentes** est un graphe dirigé G=(N,F) construit comme suit :

- Créer un nœud pour chaque transaction.
- Créer une flèche $T_i \rightarrow T_j$ quand la transaction T_i attend de verrouiller un élément actuellement verrouillé par T_j.

Un verrou indéfini existe si et seulement si le graphe des attentes contient un cycle (Holt, 1972).
Exercice : dessiner le graphe des attentes des transactions du transparent 23.

UNIVERSITÉ CÔTE D'AZUR

Récupération à la suite de la détection d'un verrou indéfini

- Choisir la victime du verrou indéfini.
  - Il est préférable d'annuler une transaction qui vient juste de démarrer qu'une autre s'exécutant déjà depuis un moment.
  - Il est préférable d'annuler une transaction qui n'a encore apporté que quelques modifications, plutôt qu'une autre qui a déjà effectué un travail important sur la base de données.
  - Il est préférable d'annuler une transaction qui a encore beaucoup de modifications à apporter qu'une autre qui n'en a plus que quelques unes à effectuer.
- Éviter la famine (starvation).