Gestion de transactions centralisée

Patricia Serrano Alvarado
Laboratoire d' Informatique de Nantes Atlantique (LINA)
Patricia.Serrano-Alvarado@univ-nantes.fr

Transaction

- Un utilisateur manipule une base en écrivant des programmes d'application qui font des appels au SGBD. L'exécution d'un programme fait naître au niveau du SGBD l'occurrence d'une transaction

- Une transaction (T) est un groupe d'actions (lectures, écritures) sur une base de données

- Les notions de cohérence et tout ce qui concerne le support transactionnel sont indépendantes du modèle de données utilisé par la base de données

Cycle de vie d'une transaction

- Une transaction peut :
  - se dérouler normalement,
  - être tuée en cours d'exécution ou
  - s'arrêter par elle même avant la fin.

- Les opérations d'une transaction
  - début
  - fin (validation, annuler)
  - lire : lecture de la valeur d'un objet à partir de la BD et stockage de la valeur dans l'espace de travail de la transaction
  - écrire : à partir d'une valeur stockée dans l'espace de travail de la transaction et écrire cette valeur dans la base pour l'objet désigné
  - abandonner (rollback, abort) : défaire toute les màj faites par la transaction depuis son début

Propriétés ACID (1)

- Atomicité, Cohérence, Isolation, Durabilité

- Atomicité
  Soit T se termine correctement et elle a les effets désirés sur les données manipulées (elle est validée)
  Soit T est interrompue et elle n'a aucun effet sur les données (elle est abandonnée)

  Propriété essentielle pour assurer la cohérence "syntaxique" de la base (suite à une panne et défaillance)

- Cohérence : les modifications faites par la transaction ne mettent pas en péril l'intégrité de la base.

  Nécessite un contrôle de l'intégrité (CI) fait en général en phase de validation. En général du ressort du programmeur.

Propriétés ACID (2)

- Isolation
  Cette propriété permet d'éviter les interférences entre T et les autres transactions, produisant des résultats incorrects.
  Tout se passe comme si la transaction s'exécutait de manière isolée des autres transactions.

  Isolation : *protocoles de contrôle de concurrence*

- Durabilité
  Le système conserve toute modification faite par une transaction qui a validé. Ceci demande des techniques de résistance aux défaillances

  Durabilité (et Atomicité) : *protocoles de reprise*

Le travail du gestionnaire des transactions

- Initialiser chaque transaction et contrôler son exécution.

  Si celle-ci se passe bien => confirmer la transaction
  Sinon annuler la transaction en défaisant ses opérations.

- Contrôler les *accès concurrents* *en synchronisant* les transactions en conflit.

- Assurer la *reprise après panne* : refaire le travail des transactions ayant atteint leur point de confirmation (commit/ validation) avant la panne et défaire celles qui n'avaient pas atteint ce point au moment de la panne.

Deux problématiques

1. Gestion des accès concurrents
   - TRANSACTION Ti = séquence d'actions (LIRE, ECRIRE)
   - Exécution concurrente de N TRANSACTIONS:
     ORDONNANCEMENT (S) dans le temps des actions (A) de ces transactions

S = < ..... (T1, A1, x) ..... (T2, A2, x) .....>

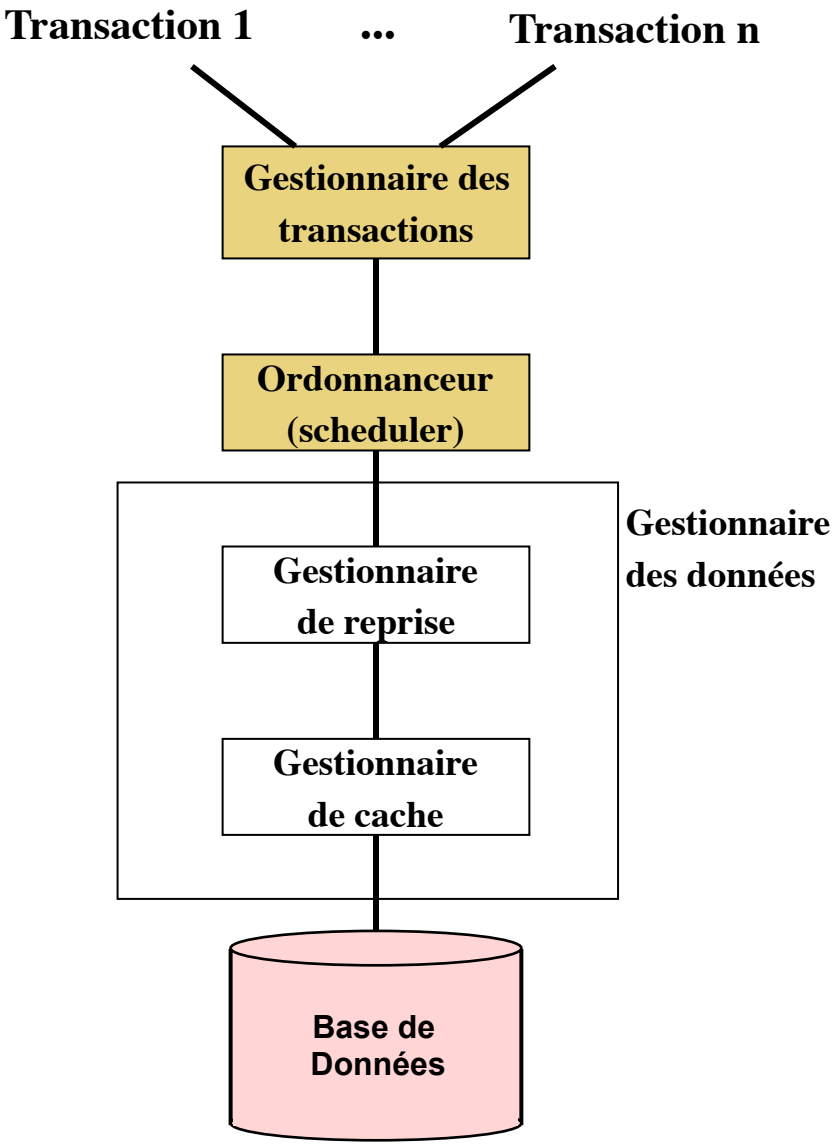
----------|----------|----------> temps

2. Reprise après pannes pour remettre la base dans un état cohérent (fiabilité)

Ordonnancements

Gestion centralisée d’accès concurrents

SGBD centralisé
Architecture abstraite



Ordonnancement séquentiel

- Par définition il s'agit d'un ordonnancement COHERENT
- Il peut y avoir plusieurs exécutions séquentielles donnant des résultats différents.
- Inconvénients : aucun parallélisme, faible débit, délais non nécessaires

Temps

S1

T1	T2	A	B
		25	25
lire(A,t)			
t:=t+100			
écrire(A,t)		125	
lire(B,t)			
t:=t+100			
écrire(B,t)			125
	lire(A,s)		
	s:=s*2		
	écrire(A,s)	250	
	lire(B,s)		
	s:=s*2		
	écrire(B,s)		250

S2

T1	T2	A	B
		25	25
	lire(A,s)		
	s:=s*2		
	écrire(A,s)	50	
	lire(B,s)		
	s:=s*2		
	écrire(B,s)		50
lire(A,t)			
t:=t+100			
écrire(A,t)		150	
lire(B,t)			
t:=t+100			
écrire(B,t)			150

Ordonnancement sérialisable

Un ordonnancement est SERIALISABLE s'il a le même effet sur la base qu'une exécution séquentielle de transactions

- Ordonnancements non séquentiels cohérents ?

- Pas de problème si les transactions travaillent sur des objets différents, mais c’ est également possible pour des transactions travaillant sur les mêmes entités.
- Les problèmes d'incohérence sont liés aux ordres de lecture/écriture ou écriture/écriture sur une même entité

Ordonnancement sérialisable

- Il peut y avoir plusieurs exécutions non séquentielles donnant des résultats différents

S3 = Sérialisable (cohérent)

T1	T2	A	B
		25	25
lire(A,t)			
t:=t+100			
écrire(A,t)		125	
	lire(A,s)		
	s:=s*2		
	écrire(A,s)	250	
lire(B,t)			
t:=t+100			
écrire(B,t)			125
	lire(B,s)		
	s:=s*2		
	écrire(B,s)		250

S4 = Sérialisable (cohérent)

T1	T2	A	B
		25	25
	lire(A,s)		
	s:=s*2		
	écrire(A,s)	50	
lire(A,t)			
t:=t+100			
écrire(A,t)		150	
	lire(B,s)		
	s:=s*2		
	écrire(B,s)		50
lire(B,t)			
t:=t+100			
écrire(B,t)			150

Ordonnancement sérialisable

S5 = Non sérialisable (non cohérent)

T1	T2	A	B
		25	25
lire(A,t)			
t:=t+100			
écrire(A,t)		125	
	lire(A,s)		
	s:=s*2		
	écrire(A,s)	250	
	lire(B,s)		
	s:=s*2		
	écrire(B,s)		50
lire(B,t)			
t:=t+100			
écrire(B,t)			150

Équivalence entre deux ordonnancements

- Équivalence basée sur le résultat

  Problème : quelque fois le résultat est accidentellement égal mais dépend de l'état initial de la base de données.

- Commutativité

  Hypothèse : les opérations sont commutatives

  Deux ordonnancements sont équivalents si ils exécutent les mêmes actions sans regarder l'ordre.

  Hypothèse trop forte, les opérations sont rarement toutes commutatives

Équivalence entre deux ordonnancements

- Deux ordonnancements S1 et S2 sont équivalents s'ils incluent exactement les mêmes transactions et l'effet final des deux est le même indépendamment de l'état initial de la base.

- Soit Ti Tj les transactions participant à deux ordonnancements S1 et S2
  1. Pour tout LIRE(X) fait par Ti dans S1, si la valeur lue est celle modifiée (en dernier) par ECRIRE(X) de Tj dans S1 alors la même opération LIRE(X) de Ti dans S2 doit lire la valeur écrite par ECRIRE(X) de Tj dans S2.

     S1 = ... Écrire Tj (X), Lire Ti (X) ...
     S2 = ... Écrire Tj (X), Lire Ti (X) ...

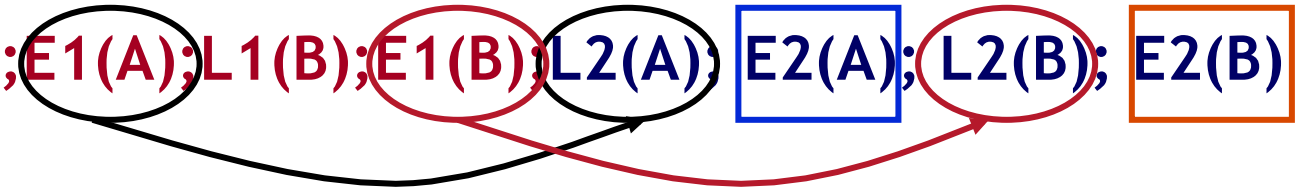
  2. Pour toute entité X sur laquelle il y a un dernier ECRIRE(X) exécutée par Ti dans S1 alors la même opération ECRIRE(X) par Ti doit être la dernière dans S2.

     S1 = ... Écrire Ti (X)
     S2 = ... Écrire Ti (X)

Équivalence entre deux ordonnancements

- Lire de Ti sur l'objet A = Li(A)
- Écrire Ti sur l'objet A = Ei(A)

S1 = L1(A);E1(A);L1(B);E1(B);L2(A); E2(A); L2(B); E2(B)



S2= L1(A);E1(A); L2(A);E2(A); L1(B);E1(B); L2(B);E2(B)

On peut conclure que S2 est sérialisable parce qu'il est équivalent à S1

Conflit-sérialisable

- Condition suffisante pour assurer qu'un ordonnancement est sérialisable

- Conditions
  - Deux ordonnancements sont conflit-équivalent si tous les deux peuvent être inter-changés l'un par l'autre par la séquence des échanges d'opérations NON conflictuelles adjacentes
  - Deux opérations de la même transaction ne peuvent pas être inter-changées

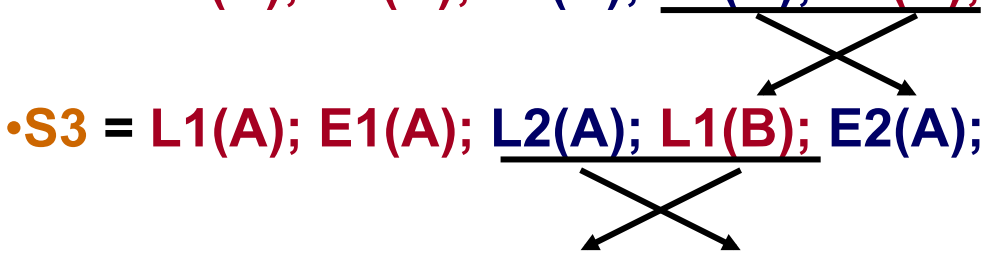
- Conflits : si T1 et T2 s'intéressent à un même objet.

- Pour un objet seules les opérations lire et écrire sont permises. Cela donne :

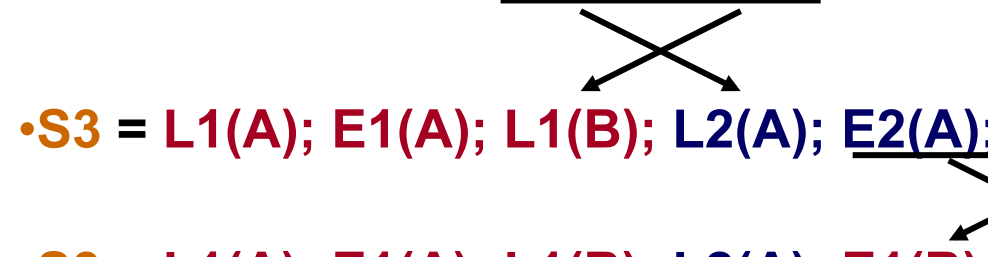
	Lecture T1(X)	Ecriture T1(X)
Lecture T2 (X)		CONFLIT
Ecriture T2(X)	CONFLIT	CONFLIT

Conflit-sérialisable

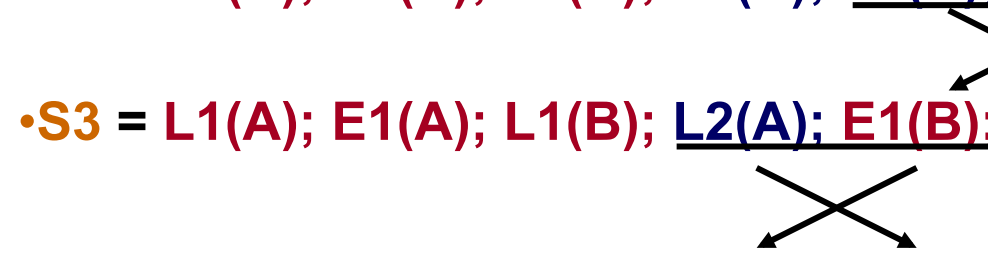
S3 = L1(A); E1(A); L2(A); E2(A); L1(B); E1(B); L2(B); E2(B)



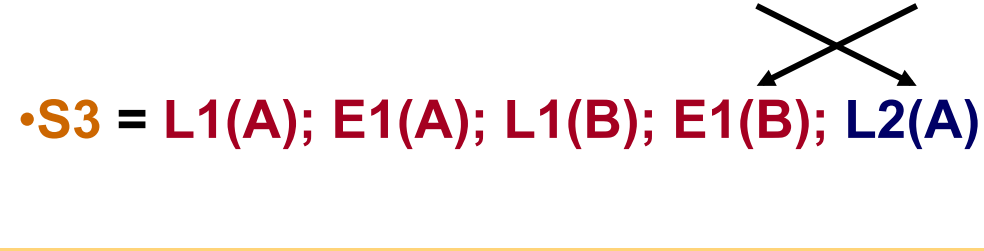
•S3 = L1(A); E1(A); L2(A); L1(B); E2(A); E1(B); L2(B); E2(B)



•S3 = L1(A); E1(A); L1(B); L2(A); E2(A); E1(B); L2(B); E2(B)



•S3 = L1(A); E1(A); L1(B); L2(A); E1(B); E2(A); L2(B); E2(B)



•S3 = L1(A); E1(A); L1(B); E1(B); L2(A); E2(A); L2(B); E2(B)

On peut conclure qu'un ordonnancement est sérialisable s'il est conflit-équivalent à un ordonnancement séquentiel

Exercice

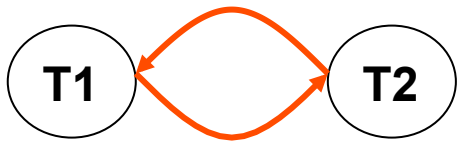
- Cet ordonnancement est-il sérialisable ?
- Dans cet exercice : r=L et w=E

S4 = r1(x) r2(y) w1(y) r3(y) w2(z) w1(x) r3(x) w3(z)

Graphes de dépendance

- Test pour vérifier si un ordonnancement est conflit - sérialisable
- Pour chaque ordonnancement S, on définit une relation de précédence entre les transactions qui indique la précédence dans le temps, d'actions conflictuelles sur un même objet
- Si le graphe est acyclique l'ordonnancement est conflit - sérialisable
- Pour construire le graphe :
  - les sommets sont les transactions
  - on a une arête de Ti vers Tj si dans l'ordonnancement un objet x est :
    - Écrit par Ti puis Écrit par Tj ou
    - Écrit par Ti puis lu par Tj ou
    - Lu par Ti puis Écrit par Tj
  - et aucune écriture (faite par une autre transaction) intervient entre les deux opérations

Conflit-sérialisable



Non conflit-sérialisable

Conflits entre transactions

Chaque transaction a au cours de son exécution et à un instant donné une image de la base qui diffère ou non de la réalité.

Reprenant les conflits possibles on identifie quelques problèmes :

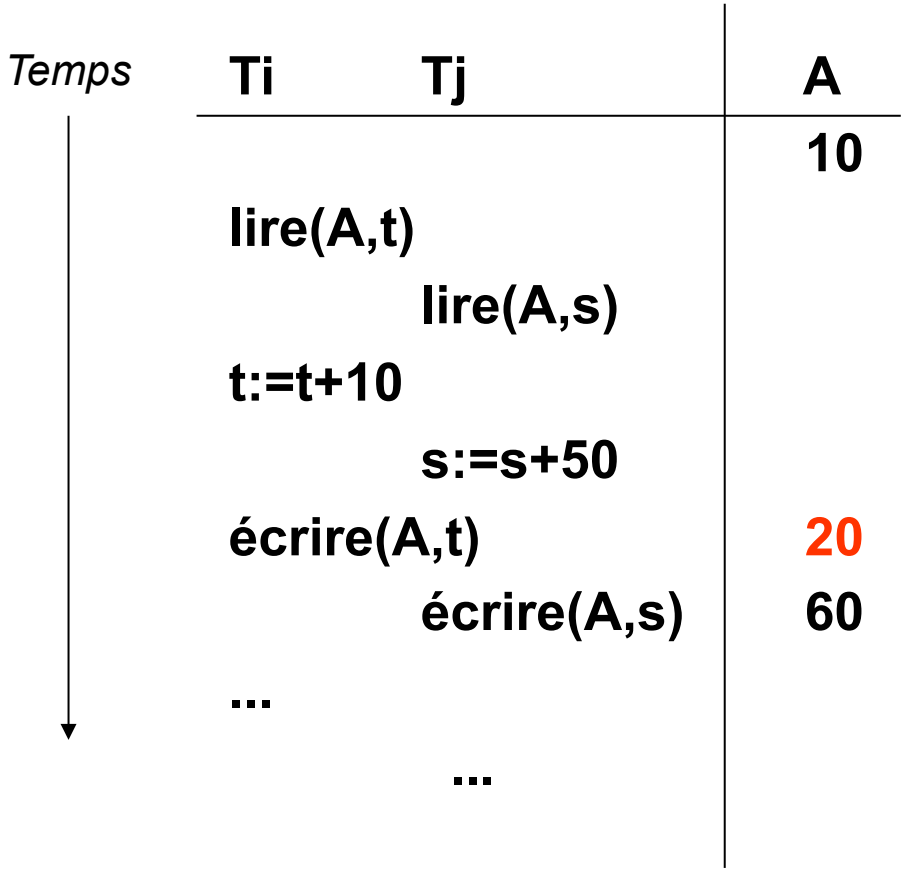
lire(A) - lire(A) et partage ;

écrire(A) - écrire(A) : perte de mise à jour ;

écrire(A) - lire(A) : lectures impropres ;

lire (A) - écrire(A) : lectures non reproductibles.

Écriture-écriture : perte de mise à jour



Écriture - lecture : lectures impropres

Ti	Tj	A
		10
lire(A,t)		
t:=t+20		
écrire(A,t)		30
	lire(A,s)	
annulation		
	...	

Lecture - écriture : lectures non reproductibles

Ti	Tj	A
		10
lire(A,t)		
	lire(A,s)	
t:=t+20		
écrire(A,t)		30
	lire(A,s)	
...		
	...	

Niveaux de cohérence

- **Niveau 1** : Pas de perte de màj
- *Toute T ne doit pas écrire sur un objet dont la valeur a été modifiée par une autre T' qui n'a pas atteint son point de confirmation.*

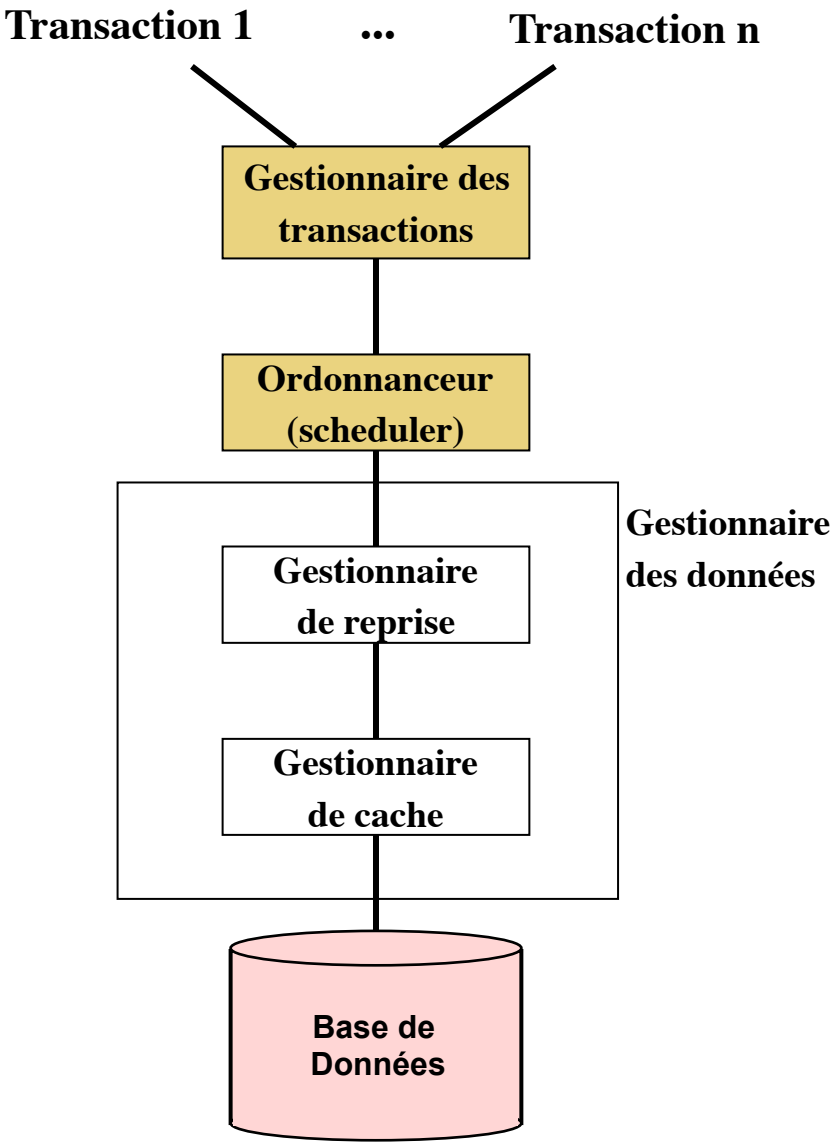
- **Niveau 2** : Ni pertes de màj ni lectures sales
- *Toute transaction T ne doit pas lire des valeurs non confirmées, manipulées par d'autres transactions.*

- **Niveau 3** : Ni perte de màj ni lectures sales et non reproductibles
- *Aucune transaction ne doit modifier une valeur lue par une autre transaction avant que cette dernière ne soit terminée.*

Comment gérer la sérialisabilité et les niveaux de cohérence ?

- Techniques pour contrôler les actions concurrentes pour assurer qu'un ordonnancement est équivalent à un ordonnancement séquentiel
- Différentes approches
- Pessimistes (ex. exclusion mutuelle, estampilles, verrouillage à deux phases)
- Optimistes (ex. méthode par certification)
- Tout d'abord on impose l'utilisation de verrous pour assurer *l'exclusion mutuelle*...

SGBD centralisé
Architecture abstraite



Protocole d'exclusion mutuelle

- Exclusion mutuelle : procéder à une allocation exclusive des objets que chaque transaction manipule.

- Principe : Avant d'effectuer une opération sur x (lire ou écrire),
  Acquérir le contrôle exclusif : verrouiller(x)
  Abandonner le contrôle : libérer(x)

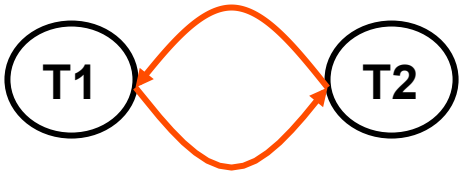
1. Règle 1 : Aucune transaction ne peut effectuer une mise à jour ou une lecture d'un objet si elle n'en a pas acquis au préalable le contrôle exclusif par verrouiller
2. Règle 2 : Si une transaction T2 ne peut acquérir le contrôle exclusif d'un objet X, parce que X est utilisé par une autre transaction T1, T2 doit attendre jusqu'à ce que X soit libéré par T1.

Permet de résoudre le problème des pertes de màj
(Niveau de cohérence 1)

Exemple d'exclusion mutuelle

T1	T2	A
		10
verrouiller(A)		
lire(A,t)		
	verrouiller(A)	
	attente	
t:=t+10		
écrire(A,t)		20
libérer(A)		
	lire(A,s)	
	s:=s+50	
	écrire(A,s)	70
	libérer(A)	

Protocole d'exclusion mutuelle: Inconvénients

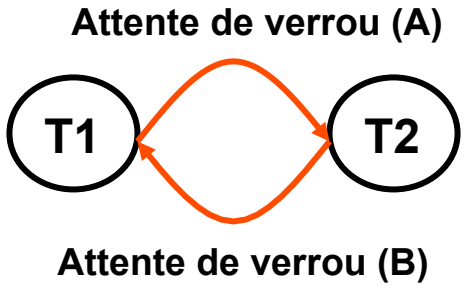
- Non suffisant pour éviter :
  - Lectures sales et non reproductibles
- Pas de garantie des niveaux de cohérence 2 et 3)
- Pas de garantie d'ordonnancements non séquentiels cohérents (conflit - serialisable), exemple :
- 

Non conflit-sérialisable = non sérialisable

T1	T2	A	B
		25	25
verrouiller(A)			
L(A,t)			
t:=t+100			
E(A,t),Lib(A)		125	
	verrouiller(A)		
	L(A,s)		
	s:=s*2		
	E(A,s), Lib(A)	250	
	verrouiller(B)		
	L(B,s)		
	s:=s*2		
	E(B,s), Lib(B)		50
verrouiller(B)			
L(B,t)			
t:=t+100			
E(B,t),Lib(B)			150

Interblocage (deadlock)

- Le verrouillage entraîne le problème suivant :



T1	T2	A	B
		25	25
verrouiller(A)			
L(A,t)			
t:=t+100			
E(A,t)		125	
	verrouiller(B)		
	L(B,s)		
	s:=s*2		
	E(B,s),		50
verrouiller(B)			
attente			
	verrouiller(A)		
	attente		

- Situation mise en évidence par un graphe Qui Attend Quoi (QAQ)

- Deux grandes techniques pour résoudre ce problème:
  - PREVENTION : éviter l'apparition de l'interblocage
  - DETECTION : laisser les interblocages se produire et les détecter

Interblocage : prévention

- **Définir un ordre des transactions**

- **Quand une transaction Tj demande un objet A détenu par Ti, on leur fait passer un test.**
- **Si le test réussit on fait attendre Tj sinon on tue l'une des transaction pour être relancée par la suite**

- **Deux possibilités :**
  - **sans préemption (tuer Tj) ou**
  - **avec préemption (tuer Ti)**

- **Test basé sur des priorités**
  - **Tj peut attendre Ti si Priorité(Tj) < Priorité(Ti)**

- **Risque : transaction malchanceuse qui attend tout le temps**

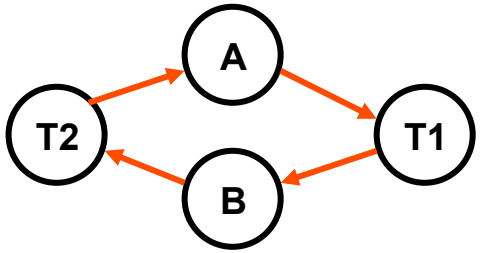
- **Pour éviter cette situation une variante : utilisation d'estampilles**

Interblocage : détection

- Construire le graphe Qui Attend Quoi (ou Qui)

  Nœuds = Transactions (et Objets)

  Arêtes = de l'objet vers Ti si Ti a verrouillé l'objet, de Ti vers l'objet si Ti attend



- Détection des cycles dans le graphe : quand examiner le graphe ?
  - transaction en attente
  - périodiquement (détection trop tard si période trop grande)
- Choix d'une transaction à annuler pour casser le cycle:
  - la transaction détenant le moins de ressources (la plus jeune)
- Relancer la transaction annulée plus tard
- En pratique, les interblocages ne sont pas fréquents (1/1000 trans.)
  - Détection plus économique
  - Mais prévention, régulateur de charge

Contrôle de concurrence avec estampilles

- A la création d'une transaction on lui affecte un numéro d'ordre unique : son estampille (ex: horloge du système)
- On utilise les estampilles pour définir priorités : les transactions les plus vieilles ont la priorité sur les plus jeunes
- Dans la prévention d'interblocage avec estampilles une transaction Tj a le droit d'attendre une transaction Ti si Tj > Ti (i.e., si Tj est plus jeune que Ti)
- Ensuite on peut appliquer une politique avec ou sans préemption.
- Sans préemption

  Tj essaie d'attendre Ti. Si Tj est plus jeune que Ti on l'autorise à attendre, sinon Tj est tuée mais on ne lui change pas son estampille. Elle est relancée un peu plus tard avec la même estampille ce qui la met dans une situation meilleure par rapport aux priorités.
- Avec préemption

  Si Tj est plus vieille que Ti elle attend sinon Ti est annulée

Verrouillage à deux phases

- **Exclusion mutuelle est trop restrictive et n'est pas suffisant pour garantir :**

1. **Ni perte de màj ni lectures sales et non reproductibles**
   (Cohérence niveau 3)
2. **Ordonnancement non séquentiel cohérent (conflit - sérialisable),**

- **Idée : pour chaque transaction tous les verrouillages doivent précéder toutes les libérations ; phase d'acquisition des verrous puis phase de libération.**

- **Pour optimiser l'accès concurrent deux types de verrous :**
  - **PARTAGEABLES en LECTURE : V- partagé (A)**
  - **EXCLUSIF en ECRITURE : V- exclusif (A)**

	V-partagé(A)	V-exclusif(A)
V-partagé(A)		CONFLIT
V-exclusif(A)	CONFLIT	CONFLIT

Protocole de verrouillage à deux phases (2PL)

- Principe :

  Transaction t *bien formée* si :

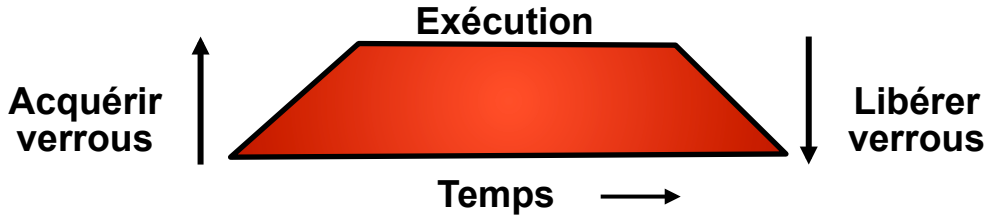
  i. Avant de LIRE x elle a au moins un V-partagé (x)

  ii. Avant d'ECRIRE x elle a un V-exclusif (x)

  iii. Aucun objet ne reste verrouillé après FIN TRANSATION T

  Transaction à deux phases (2 Phase Locking) :

  - Règle I : Des transactions différentes ne peuvent posséder simultanément des verrous en conflit
  - Règle II : Toute transaction qui libère un verrou ne peut plus en acquérir d'autres



- Théorème : Si dans un ordonnancement toutes les transactions sont à deux phases et bien formées alors tout ordonnancement est équivalent à un ordonnancement séquentiel (qui est donc cohérent)

Exemple

- Quelle est l'exécution obtenue par verrouillage à deux phases à partir de l'ordonnancement S5 ?
- On considère que lorsque les verrous sont libérés (au moment du commit/ terminaison) on exécute en priorité les opérations bloquées dans l'ordre de leur blocage

- S5 = L1(A); E1(A); L2(A); E2(A); L1(B); E1(B); L2(B); E2(B); C1;C2

Attend	Tient verrou	VL	VE
	T1	X	
	T1		X
T2		A	
T2			A
	T1	X	
	T1		X
	T2	X	
	T2		X
	T2	X	
	T2		X
	C1		
	C2		

Ordonnancements non recouvrables

- 2PL assure des ordonnancements conflit-sérialisables
- Mais n'évite pas les ordonnancements non recouvrables
  - Considerez

    Libération des verrous

    S6 = L1(A); E1(A); L2(A); E2(A); L2(B); E2(B); **C2; A1**

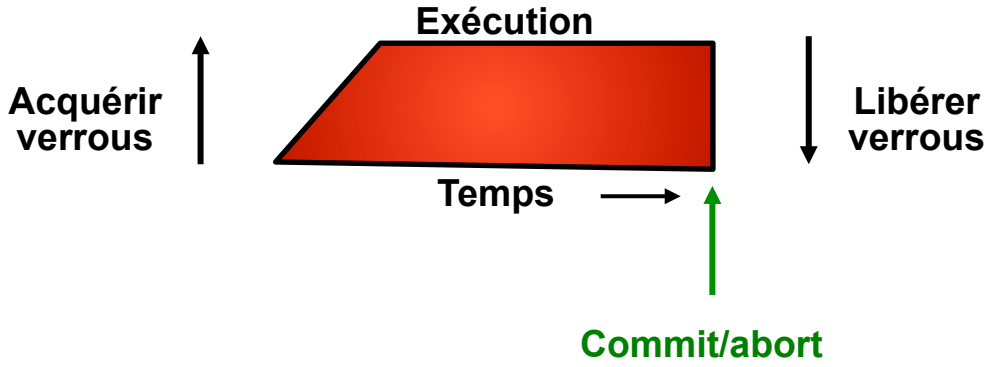
    Libération des verrous

    C2= commit de T2

    A1=abort de T1
  - C2 a validé et ne peut plus faire marche en arrière !

Protocole de verrouillage à deux phases strict (S-2PL)

- S-2PL en plus d'assurer des ordonnancements conflit-sérialisables assure des ordonnancements recouvrables
- Dans un ordonnancement recouvrable les transactions valident uniquement après que toutes les transactions dont elles ont lu les modifications (conflit écriture-lecture) ont validé
- La règle II de 2PL est remplacée par :
  - Tous les verrous sont libérés à la fin de la transaction (commit/ abort)



Graphe de dépendances avec verrous exclusifs et partagés

Construction du graphe :

- On a une arête de Ti vers Tj si Ti fait VL(x) ou VE(x) et si Tj est la transaction suivante à verrouiller x en écriture VE(x)

- Si Ti relâche un verrou exclusif sur x alors il y a une arête de Ti vers toutes les transactions Tm qui posent un verrou en lecture VL(x) sur x jusqu'à ce qu'une transaction pose un verrou en écriture sur x.

- S'il n'a pas de cycle on peut produire un ordonnancement séquentiel équivalent (l'ordonnancement est sérialisable)

Protocole de verrouillage à deux phases strict

- Garantit :
  - Niveau de cohérence 3
  - Sérialisabilité
- Mais, problème de fantômes ?

Le problème des fantômes

Objets fantômes : objets apparaissant/disparaissant d'un ensemble d'objets sélectionnés

Relation EMP(NE, DPT, SAL)

NE	DPT	SAL
e1	d1	50
e2	d1	45
e3	d2	25
e4	d2	30
e5	d2	48
e6	*d1*	*60*

Relation GENERALE(DPT, MAXSAL)

DPT	MAXSAL
d1	50
d2	48

Transactions T1 et T2

	T1
A1	SELECT MAX(SAL) INTO tt FROM EMP WHERE DPT=d1
A2	UPDATE GENERALE SET MAXSAL = tt WHERE DPT = d1

	T2
B1	INSERT INTO EMP VALUES(e6,d1,60)

Comment éviter l'insertion d'employés après la sélection ?

Fantômes (cont.)

NE	DPT	SAL
e1	d1	50
e2	d1	45
e3	d2	25
e4	d2	30
e5	d2	48
e6	*d1*	*60*

	T1
A1	SELECT MAX(SAL) INTO tt FROM EMP WHERE DPT=d1
A2	UPDATE GENERALE SET MAX-SAL = tt WHERE DPT = d1

	T2
B1	INSERT INTO EMP VALUES(e6,d1,60)

	Exécution 1	Exécution 2	Exécution 3
1	A1	B1	A1
2	A2	A1	B1
3	B1	A2	A2
	max d1 : 50	**max d1 : 60**	**max d1 : 50**
	max d2 : 48	**max d2 : 48**	**max d2 : 48**

- **Exécution 3 non sérialisable, non cohérente**
- **T1 ne peut pas verrouiller des n-uplets n'existant pas**
- **Verrou des n-uplets ne suffit pas : Verrou sur toute la relation pour éviter insertions et suppressions de n-uplets**
- **Plus généralement : verrouillage par prédicat -- la demande de verrou spécifie l'ensemble d'entités à verrouiller**

Autres techniques

- Protocoles optimistes par certification
- Verrouillage hiérarchique
- Transactions distribuées

SQL et Transactions

- Début de transaction implicite (une instruction SQL)
- Fin de transaction
  - implicite (après chaque instruction, exemple "autocommit off")
  - explicite : COMMIT pour valider, ROLLBACK pour annuler
- Type et degré d'isolation d'une transaction
  - Actions explicites de verrouillage et libération (Lock, Unlock)
  - Granule de verrouillage ? tuple, relation, index, page, segment
  - Verrouillage hiérarchique (intentions)
- SQL2 : SET TRANSACTION <mode> <isolation> ...
- MODE :
  - READ ONLY : lecture seulement, pas de mise à jour de la BD
  - READ WRITE
- ISOLATION LEVEL
- NB pas de perte de mise à jour

Niveau	Lecture impropre	Lecture non reproductible	Fantômes
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	NON	Possible	Possible
REPEATABLE READ	NON	NON	Possible
SERIALIZABLE	NON	NON	NON

SQL2 : Isolation Level

Niveau	Lecture impropre	Lecture non reproductible	Fantômes
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	NON	Possible	Possible
REPEATABLE READ	NON	NON	Possible
SERIALIZABLE	NON	NON	NON

- Sérialisable : T verrouille au début et libère à la fin, 2PL strict.
  - Verrous sur ensembles d'objets pour éviter les fantômes.
- Repeatable read : T lit seulement les modifications de transactions ayant validé (commit). Aucun objet lu ou modifié par T n'est modifié par une autre transaction.
  - Verrou sur objet individuel et non sur collection, ce qui peut entraîner des fantômes.
- Read Committed : T lit seulement les mises à jour de transactions ayant validé (commit). Aucun objet modifié par T n'est modifié par une autre transaction. Mais un objet lu par T peut avoir été modifié par une autre transaction.
  - Verrous exclusifs pour écrire et ne pas les libérer avant la fin,
  - Verrous de lecture libérés de suite.
- Read Uncommitted : T peut lire des objets modifiés par une autre transaction.
  - Pas de verrous en lecture et mode "Read Only".
- Exemples :
  - SET TRANSACTION READ ONLY ISOLATION LEVEL SERIALIZABLE
  - SET TRANSACTION READ WRITE ISOLATION LEVEL READ COMMITTED

Gestion des transactions sous Oracle

Transaction en Oracle

- Démarrage : une instruction SQL exécutable démarre automatiquement une transaction
- Fin : une transaction termine soit
  - explicitement avec COMMIT ou ROLLBACK
  - implicitement avec une instruction DDL
  - Implicitement lors d'une déconnection de l'utilisateur
- Avant le COMMIT d'une transaction, toutes ses instructions peuvent être annulées avec ROLLBACK
- Uniquement après le COMMIT d'une transaction les changements sont visibles aux autres utilisateurs.

En pratique, au départ d'une transaction on fait COMMIT pour s'assurer qu'aucune transaction est active.

Niveaux d'isolation

- Oracle permet de définir le « mode » (lecture seule ou lecture écriture) d'une transaction
  - SET TRANSACTION { READ ONLY | READ WRITE };
- Si READ WRITE : 2 niveaux d'isolation
  - READ COMMITTED (SQL2) <-niveau par défaut
  - SERIALIZABLE (SQL2)
- Il est possible de spécifier le niveau d'isolation
  - au niveau de la transaction

SET TRANSACTION ISOLATION LEVEL { SERIALIZABLE | READ COMMITTED };

  - au niveau de la session

ALTER SESSION SET ISOLATION_LEVEL { SERIALIZABLE | READ COMMITTED };

Choix d'un niveau d'isolation

- READ COMMITTED
  - pour les environnements où peu de transactions risquent d'entrer en conflit.
  - les lectures non répétables et fantômes sont possible, mais le temps d'exécution est meilleur.
- SERIALIZABLE (plus long),
  - pour les environnements avec de grosses BD et des transactions courtes qui mettent à jour peu de lignes.
  - où les transactions longues concernent surtout des lectures.

Points de sauvegarde

- SAVEPOINT
  - définition de «points de sauvegarde» au sein d'une transaction qui permettent d'organiser la transaction en plus petites unités notamment pour le ROLLBACK.
- ROLLBACK TO SAVEPOINT <sp>
  - ne concerne que les opérations effectuées après le point <sp> ;
  - préserve <sp> mais perd tous les points de sauvegarde définis après <sp> ;
  - relâche tous les verrous (table ou ligne) posés depuis le point <sp> ;
  - La transaction continue

Suite

- En plus des commandes précédentes, gérer soi-même les verrous
  - LOCK TABLE IN ROW EXCLUSIVE MODE
  - ROW SHARE MODE
  - SHARE MODE
  - EXCLUSIVE MODE...
  - (Voir les cours de M1 et M2 informatique)

Bibliographie

- Database systems. The complete book, Second edition. Hector Garcia Molina, Jeffrey D. Ullman, Jennifer Widom. Pearson International Edition, 2009.
- Database management systems.Third edition. Raghu Ramakrishnan, Johannes Gehrke. Mc Graw Hill, 2003.
- An introduction to database systems. C. Date. Addison Wesley, 1986.
- Documentation Oracle http://docs.oracle.com/cd/B28359_01/server.111/b28318/transact.htm