



# Gestion des transactions et reprise après pannes

---



# Références

---

- Concurrency Control and Recovery in Database Systems. Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman
- Database systems. The complete book. H. Garcia-Molina, J-D. Ullman, J.Widom



# Plan

---

- Notions de base
- Propriétés ACID
- Réparation des exécutions
- Théorie de la sérialisabilité



# Problèmes étudiés

---

- Contrôle de la concurrence
  - Activité de coordination de processus qui opèrent en parallèle, **accèdent à des données partagées**, et peuvent ainsi potentiellement **interférer**
- Reprise après pannes
  - Activité qui garantit que les pannes logicielles et matérielles **n'altèrent pas** les données **persistantes**
- On retrouve ces problèmes dans différents domaines
  - Conception du matériel
  - Système d'exploitation
  - Systèmes temps réel
  - Systèmes distribués
  - **SGBDs**



# Exemple (1/3)

Client(cnum, nom, adr, solde)  
Réservation(vnum, date, cnum, spec)  
Vol(vnum, date, dep, dest, placeV, cap)  
**Contrainte** :  $placeV \leq cap$

Begin

...  
**update Vol set placeV:=placeV + 1**  
**where vnum=x.vol and date=x.date;**



**coupure d'électricité**

...  
**insert into Réservation values (x.vol, x.date, x.cnum, null);**

...

end;

## Problème 1:

**Que se passe-t-il s'il n'y a plus de places disponibles dans le vol considéré ?**

## Problème 2 :

**Que se passe-t-il s'il y a une panne après l'incrémentation du nombre de places vendues ?**



## Exemple (2/3)

**Begin**

```
...  
    update Vol set placeV:=placeV +1  
    where vnum='001' and  
date='1/12/08';  
...  
...  
    insert into Réservation  
    values ('001','1/12/08', x.client,  
null);  
...  
end;
```

**Begin**

```
...  
select placeV into y from Vol  
where vnum= vnum='001' and date='1/12/08';  
...  
y = y+1;  
..  
update Vol set placeV:=y  
where vnum=x.vol and date=x.date;  
...  
insert into Réservation  
    values ('001','1/12/08', x.client, null);  
...  
end;
```

## Exemple (3/3)

Temps

**select placeV into y from Vol  
where vnum='001' and date='1/12/08';**

**update Vol set placeV:=placeV +1  
where vnum='001' and date='1/12/08';**

**y = y+1;  
update Vol set placeV:=y  
where vnum='001' and date='1/12/08';**

**insert into Réservation  
values ('001', '1/12/08', x.client, null);**

**insert into Réservation  
values ('001', '1/12/08', x.client, null);**

### Problème:

**Le processus rouge  
utilise une valeur de la  
variable x qui est  
devenue obsolète**



# Qu'est-ce qu'une transaction ?

---

Le concept de requête ne permet pas de capturer les notions d'exécution consistante ou de traitement fiable

## → Notion de transaction

**“Execution d' un programme qui accède à une base de données”**

Du point de vue du SGBD : c'est une séquence d'operations de lecture et écriture sur les données

## Exemples

- Réservation et/ou achat d'un billet d'avion
- Commande d'un produit sur un site internet
- ...





# Consistance et fiabilité

---

Transaction: unité de base pour un traitement consistant et fiable

## ■ Deux notions de consistance

### ■ Consistance d'une base de données

- un état de base de données qui vérifie toutes les contraintes d'intégrité définies sur la base de données

### ■ Consistance d'une transaction

- liée aux actions de transactions concurrentes
- maintenir la consistance d'une base de données lors des accès concurrents

## ■ Notion de fiabilité

- tolérance aux différents types de pannes
- capacité de reprise après une panne



# Gestion des transactions

---

Comment garder en permanence la base de données dans un état consistant même lors des accès concurrents ou de pannes ?

## ■ Objectifs

- Transparence de la concurrence
- Transparence aux échecs

## ■ Pourquoi c'est difficile ?

- Fiabilité
- Disponibilité
- Temps de réponse – entre 1-2 seconds
- Débit – milliers de transactions/second
- Passage à l'échelle – échelle d'Internet
- Sécurité
- Configurabilité
- Atomicité
- Durabilité
- Distribution

# Propriétés, opérations et états

## ■ Une transaction doit vérifier les propriétés **ACID**

### ■ **A**tomacité

Tout ou rien

### ■ **C**ohérence

Maintenir l'intégrité de la base de données

### ■ **I**solation

Transparence de la concurrence

### ■ **D**urabilité

Les modifications effectuées par une transaction sont permanentes

## ■ Opérations

### ■ **Start**

Début de transaction

### ■ **Commit**

Validation de la transaction

### ■ **Abort**

Annulation de la transaction

**Transactions non validées (uncommitted)**

## ■ Etats

### ■ Validé (**committed**)

Terminée par un commit

### ■ **Active**

Toujours en cours d'exécution

### ■ Annulée (**aborted**)

Terminée par un abort



# Durabilité

---

- Les effets des transactions validées sont persistants
  - ... Une transaction peut donc être vue comme un contrat
- Elle est garantie par le mécanisme de **journalisation**
  - Journal = un fichier en mémoire persistante
  - Le SGBD consigne toutes les modifications d'une transaction dans un journal
  - Les ordres commit sont également journalisés
  - Une transaction est effectivement validée (e.g., message 'commit complete' envoyé à l'utilisateur) uniquement après la journalisation du commit



# Atomicité

---

- Une transaction est une unité d'exécution
  - Deux terminaisons possibles
    - Succès (validation) : **commit**
    - Echec (annulation) : **abort**
  - Pas de validation partielle de la transaction
- L'opération **Abort** **défait** les opérations déjà exécutées
  - On dit aussi qu'elle fait un **Rollback**
  - Permet la restauration des valeurs des données d'avant le début de la transaction
  - Différentes raisons peuvent conduire à une annulation de la transaction
    - Panne du système
    - Annulation par le SGBD
      - Le système ne peut pas garantir l'atomicité
      - Violation d'une contrainte d'intégrité de la BD
      - Problème d'isolation de l'exécution
      - Ressource non disponible
  - La transaction exécute un rollback

# Exemple (1/2)

**Rollback**

## Begin Transaction T1

```
...  
update Vol set placeV:=placeV +1  
where vnum=x.vol and date=x.date;  
...  
...  
insert into Réservation  
values (x.vol, x.date, x.client, null);  
...
```

**End Transaction T1;**

**<start-T1>**

...

```
update Vol set placeV:=placeV +1  
where vnum= '001' and date='1/12/08';
```

...

**panne**

**Abort**

Temps

## Exemple (2/2)

**Violation de la contrainte  
 $\text{placeV} \leq \text{cap}$**

**Rollback**

**<start-T1>**

...  
update Vol set placeV:=placeV +1  
where vnum= '001' and date='1/12/08';  
...

**Abort**

**Begin Transaction T1**

...  
update Vol set placeV:=placeV +1  
where vnum=x.vol and date=x.date;  
...  
...  
insert into Réservation  
values (x.vol, x.date, x.client, null);  
...

**End Transaction T1;**

Temps



# Révocabilité des opérations

---

- Commit et abort sont **irrévocables**
- Opérations d'écriture
  - **Révocables** tant que la transaction ne s'est pas terminée
  - **Irrévocables** une fois la transaction validée (après le commit)
- Mécanisme de révocation : rollback
  - Généralement basée sur le mécanisme de **journalisation**
    - Journalisation des données
    - Journalisation des **images avant** des données
- Attention : certaines opérations ne peuvent pas être annulées
  - Exemples : impression à l'écran, envoi d'une page web sur internet, retrait d'argent, communication avec un services web, lancement d'une fusée...
  - Dans ce cas, essayer de **compenser** à défaut d'annuler





# Etats d'une transaction

---

- Reprise des transactions : il faut garder la trace des états suivants :

- Begin-transaction
- Read et Write
- End-transaction
- Commit\_transaction
- Rollback\_transaction

- Notion de journal

Garder (sur disque) la trace des opérations d'une transaction qui affectent les données d'une BD

- [start\_transaction,T]
- [,write\_itemT,X,old\_value,new\_value]
- [,read\_item,T,X]
- [commit,T]
- [,abort,T]
  
- Checkpoints



# Cohérence (1/3)

---

- Consistance d'une BD
  - État consistant = état qui ne viole pas les contraintes d'intégrité
  - BD consistante = BD dans un état consistant
- Une transaction garantit la consistance de la base de données
- Des états (temporaires) d'inconsistance sont tolérés à l'intérieur d'une transaction
  - Exemple : lors d'une validation différées des contraintes
    - Sous Oracle

**set constraint dep\_fk deferred;**

Begin tran T1

...

...

End tran T1

La vérification de la contrainte **dep\_fk** sera effectuée au moment du commit de la transaction T1

- L'exécution séquentielle (en série) de plusieurs transactions préserve la cohérence d'une BD

... Mais les **exécutions entrelacées** pas toujours



## Cohérence (2/3)

**Contrainte** :  $\text{placeV} \leq \text{cap}$

**Begin T1**

select cap into x from Vol  
where vnum= vnum='001'  
and date='1/12/08';

update Vol set placeV:=x-1  
where vnum='001' and  
date='1/12/08';

**End T1;**

**Begin T2**

select placeV, cap into p, c from Vol  
where vnum= vnum='001' and date='1/12/08';

If (c > p + 1) then p= p+1;

update Vol set placeV:=p  
where vnum=x.vol and date=x.date;

**End T2;**

# Cohérence (3/3)

## Risque d'inconsistance lors d'exécutions entrelacées

**select cap into x from Vol where vnum=  
vnum='001' and date='1/12/08';**

**update Vol set placeV:=x-1 where  
vnum='001' and date='1/12/08';**

**select placeV, cap into p, c from Vol  
where vnum= vnum='001' and  
date='1/12/08';**

**If (c > p +1) then p= p+1;**

**update Vol set placeV:=p where  
vnum=x.vol and date=x.date;**

Exécution consistante

**select placeV, cap into p, c from Vol  
where vnum= vnum='001' and  
date='1/12/08';**

**select cap into x from Vol where vnum=  
vnum='001' and  
date='1/12/08';**  
**update Vol set placeV:=x-1 where  
vnum='001' and date='1/12/08';**

**If (c > p +1) then p= p+1;**

**update Vol set placeV:=p where  
vnum=x.vol and date=x.date;**

Exécution inconsistante



# Isolation

---

- Une transaction ne doit pas voir les autres transactions
- Une transaction ne doit pas révéler ses modifications aux autres transactions tant qu'elle n'a pas été validée

## Pourquoi ?

- Sinon, cela pose beaucoup de problèmes lors d'exécutions simultanées de plusieurs transactions
- On ne veut pas que ces problèmes soient gérés par le programmeur

## Contrôle de la concurrence

- Garantir la Cohérence et l'Isolation des transactions (donc la transparence de la concurrence)
- Maintenir un degré de concurrence élevé



# Notation

---

- On considère deux opérations sur la BD
  - **Read(X)** : lire l'item x
  - **Write(X)** : modifier l'item x(pour les exemples on utilisera aussi **Write(x, val)** : mettre la valeur de x à val)
- On note aussi
  - **ri(x)** = Read(x) par la transaction  $T_i$
  - **wi(x)** = Write(x) par la transaction  $T_i$  (ou **wi(x, val)**)
  - **ci** = Commit par la transaction  $T_i$
  - **ai** = Abort par transaction  $T_i$
- Une **exécution** (ou **plan** ou **historique**)
  - Une séquence d'opérations dans l'ordre dans lequel elles ont été traitées par le SGBD
- **Exécution série** (séquentielle)
  - Plan qui n'entrelace pas les opérations des différentes transactions

# Exemple (1/3)

**T1**

read(X)  
x:= x+1  
write(x)  
commit

**T2**

read(X)  
x:= x+1  
write(x)  
commit

- Exécution en série
  - T1 puis T2 : **r1(x)w1(x)c1r2(x)w2(x)c2**

- Exécution en parallèle

- T1: read(X)
- T1: x:= x+1
- T2: read(X)
- T1: write(x)
- T2: x:= x+1
- T2: write(x)
- T1: commit
- T2: commit

**Problème**  
**Pertes de mises à jour**

**Plan vu par le SGBD :**

**r1(x) r2(x) w1(x) w2(x) c1c2**

Mise à jour de T1 écrasée par la mise  
à jour de T2

## Exemple (2/3)

**T1**

read(X)  
x:= x+1  
write(x)  
commit

**T2**

read(X)  
x:= x+1  
write(x)  
commit

### ■ Exécution en parallèle

- T1: read(X)
- T1: x:= x+1
- T1: write(x)
- T2: read(X)
- **T1: abort**
- T2: x:= x+1
- T2: write(x)
- T2: commit

### Problème

**T2 utilise une valeur de x  
obsolète**

**Résultat : un état inconsistant**

**r1(x) w1(x) r2(x) a1w2(x) c2**

**Solution : annuler T2**

**r1(x) w1(x) r2(x) a1a2**

**Oui, mais ...**



# Exemple (3/3)

## T1

read(X)  
x:= x+1  
write(x)  
commit

## T2

read(X)  
read(y)  
write(y)  
x:= x+1  
write(x)  
commit

## T3

read(y)  
y:= y\*1.06  
write(y)  
commit

### ■ Exécution en parallèle

- T1: read(X)
- T1: x:= x+1
- T2: read(y)
- T2: write(y)
- T1: write(x)
- T3: read(y)
- T2: read(X)
- T1: abort

r1(x) r2(y)w2(y) w1(x) r3(y)r2(x) a1 **a2** **a3**

- Annuler T2 pour maintenir la consistance
- Annuler T3 pour maintenir la consistance

### Problème

Annulation en cascade



# Réparation des exécutions (1/3)

---

- Généralement, ce qu'on appelle un système de reprise (recovery system) est le mécanisme qui garantit qu'une BD ne contient en définitif que les effets des **transactions validées** et aucun effet provenant de transactions non validées
- Pour cela, il faut gérer le problème de l'annulation de transactions  
Cela veut dire :
  - Être capable de défaire les effets des transactions annulées
  - Et aussi, gérer les problèmes de concurrence sous-jacents



## Réparation des exécutions (2/3)

### ■ Exemple

**r1(x) r2(y)w2(y) w1(x) r3(y)r2(x) w3(x)**

- Si T1 est annulée, il faut également annuler T2 et T3 (**annulation en cascade**)
- Mais que faire si T3 demande une validation ?
  - Risque de conflit avec la propriété de durabilité des transactions
  - T3 ne doit pas être validée jusqu'à ce que T2 soit validée (idem pour T2 vis-à-vis de T1)



# Réparation des exécutions (3/3)

Plus généralement :

- Condition de réparation (**recoverability**) d'une exécution

Une exécution P peut-être réparée (recoverable) si pour toute transaction T de P qui est validée, la validation (commit) de T suit les validations (commit) de toutes les transactions qui ont modifié des données lues par T et qui n'ont pas été annulées avant que T ne lise leurs modifications ... ouf !!

→ Cette condition permet de garantir qu'une transaction peut-être annulée sans changer la sémantique des transactions qui ont déjà été validées



## Exercice

---

- Parmi les exécutions suivantes, indiquer celles qui peuvent être réparées :
  - $w1(x) r2(x) w2(y) c2$
  - $w1(x)r1(x) w2(y) a1r2(x)w2(x)c2$
  - $w1(x)r1(x) w2(y) r2(x) a1w2(x)c2$
  - $w1(x) w3(y)r2(x) w2(y) c2c3$
  - $w1(x) w2(y)r1(y) r2(x)c1$
  - $w1(x) w2(y)r1(y) r2(x)c2$



# Retour sur l'annulation en cascade

- La condition de réparation ne suffit pas pour éviter l'annulation en cascade

Exemple :  $w1(x)r2(x)w2(y)a1$

Exécution récupérable mais qui ne le sera plus si T2 est validée

→ T2 doit être annulée

- Problèmes des annulations en cascade

- Complexité de la gestion
- La progression du système dans l'exécution des transactions n'est pas garantie

- Comment les éviter ?

(**Cond 1**) : s'assurer que toute transaction lit uniquement les valeurs modifiées par des transactions qui ont été validées

➡ Les exécutions qui vérifient cette condition sont récupérables



# Notion d'exécution stricte (1/2)

Problème : comment défaire les opérations des transactions qui ont été annulées ?

Généralement, les SGBDs utilisent l'**image-avant** de la BD

Exemple :  $w_1(x, 1) w_1(y, 3) w_2(y, 1) c_1 r_2(x) a_2$

- Exécution après annulation de T2 :  $w_1(x, 1) w_1(y, 3) c_1$

- Après l'annulation de  $w_2(y, 1)$ , on a  $y=3$

... Mais cela ne marche pas toujours

- Un autre exemple :

- Initialement :  $x=1$

- Exécution :  $w_1(x, 2) w_2(x, 3) a_1$

- Après annulation de T1 :  $x=1$  au lieu de  $x=3$

- Exécution :  $w_1(x, 2) w_2(x, 3) a_1 a_2$

- Après annulation de T1 puis de T2 :  $x=3$  au lieu de  $x=1$



## Notion d'exécution stricte (2/2)

- Pour éviter les problèmes précédents

**(Cond 2)** : s'assurer que l'exécution d'un `Write(x, val)` est repoussée jusqu'à ce que toutes les transactions qui ont préalablement modifié `x` soient validées ou annulées

- **Exécution stricte** : exécution qui vérifie **(Cond 1)** et **(Cond 2)**

i.e., exécutions qui repoussent les exécutions des lectures et des écritures d'un item `x` jusqu'à ce que toutes les transactions qui ont préalablement modifié `x` soient validées ou annulées

- Les exécution strictes évitent l'annulation en cascade et sont récupérables





# Contrôle de la concurrence

---

- Problèmes engendrés par les exécutions entrelacées de transactions
  - Exemple : problème de pertes de mises à jour
  - Mais aussi d'autres problèmes
    - Lectures impropres :  $r1(x)w1(x)r2(x)w2(x)a1$
    - Lectures non répétitives :  $r1(x)w1(y)w2(x)r1(x)$
    - ...
- Objectif du contrôle de la concurrence
  - Permettre la concurrence des transactions tout en évitant les erreurs dues aux interférences entre les exécutions

**Comment reconnaître (définir) une exécution correcte ?**



# Théorie de la sérialisabilité (1/4)

---

- La théorie de la sérialisabilité est un outil mathématique qui permet de prouver qu'une exécution est correcte ou incorrecte
- Qu'est-ce qu'une exécution correcte ?
  - Celle qui n'engendre pas d'erreurs dues à l'interférence des transactions
  - .. Mais comment les caractériser
    - L'exécution d'une **seule transaction** est correcte
    - **Les exécutions en série sont correctes**
- Exécutions sérialisables
  - Exécutions qui ont le même effet sur la BD qu'une exécution en série
  - Les exécutions sérialisables **sont correctes**



## Théorie de la sérialisabilité (2/4)

---

- Comment reconnaître les exécutions sérialisables ?
  - Problème de ré-ordonnancement des opérations à l'intérieur d'une exécution
- Notion de conflit
  - opérations conflictuelles
  - $oi(x)$  et  $oj(x)$  sont conflictuelles si  $oi = \text{write}$  ou  $oj = \text{write}$ 
    - si  $i \neq j$  (deux transactions différentes) : transactions conflictuelles
    - existence d'un conflit entre deux opérations  $\Rightarrow$  l'ordre dans lequel elles s'exécutent est important
- Les opérations non-conflictuelles peuvent être **réordonnées sans changer les effets** d'une exécution



# Théorie de la sérialisabilité (3/4)

- Notion d'**équivalence** d'exécution (**conflict-equivalence**)
  - Même ordre relatif d'exécution pour toutes paires d'opérations conflictuelles

- Exemple

- Exécutions non équivalentes

P1 : w2(x) w2(y) r2(z) C2 r1(x) **w1(x)** C1 **r3(x)** r3(y) r3(z), C3

P2 : w2(x) r1(x) **r3(x)** **w1(x)** C1 w2(y) r3(y) r2(z) C2 r3(z) C3

- Exécutions équivalentes

P1: w2(x) w2(y) r2(z) C2 r1(x) w1(x), C1, r3(x), r3(y), r3(z), C3

P3 : w2(x) r1(x) w1(x) C1 r3(x) w2(y) r3(y) r2(z) C2 r3(z) C3



## Théorie de la sérialisabilité (4/4)

- Exécutions sérialisables (conflict-serializable)
  - Exécution équivalentes (conflict-equivalent) à une exécution en série
  - Exemple

P1: w2(x) r1(x) w1(x) C1 r3(x) w2(y) r3(y) r2(z) C2 r3(z) C3

P2 : w2(x) r1(x) r3(x) w1(x) C1 w2(y) r3(y) r2(z) C2 r3(z) C3

→ P1 est conflict-sérialisable

Conflict-équivalent à

P3: w2(x) w2(y) r2(z) C2 r1(x) w1(x), C1, r3(x), r3(y), r3(z), C3

Est-ce que P2 est conflict-sérialisable ?



# Exemple

---

$r1(a)w1(a)r2(a)w2(a)r1(b)w1(b)r2(b)w2(b)$

un plan sérialisable

$r1(a)w1(a)r2(a) \text{ } r1(b) \text{ } w2(a) \text{ } w1(b)r2(b)w2(b)$

$r1(a)w1(a) \text{ } r1(b) \text{ } r2(a) \text{ } w2(a) \text{ } w1(b)r2(b)w2(b)$

$r1(a)w1(a) \text{ } r1(b) \text{ } r2(a) \text{ } w1(b) \text{ } w2(a)r2(b)w2(b)$

$r1(a)w1(a) \text{ } r1(b) \text{ } w1(b) \text{ } r2(a) \text{ } w2(a)r2(b)w2(b)$

plan série équivalent



## Test de sérialisabilité (1/2)

---

- Relation de précédence dans une exécution

E: une exécution E

T1 et T2 : deux transactions

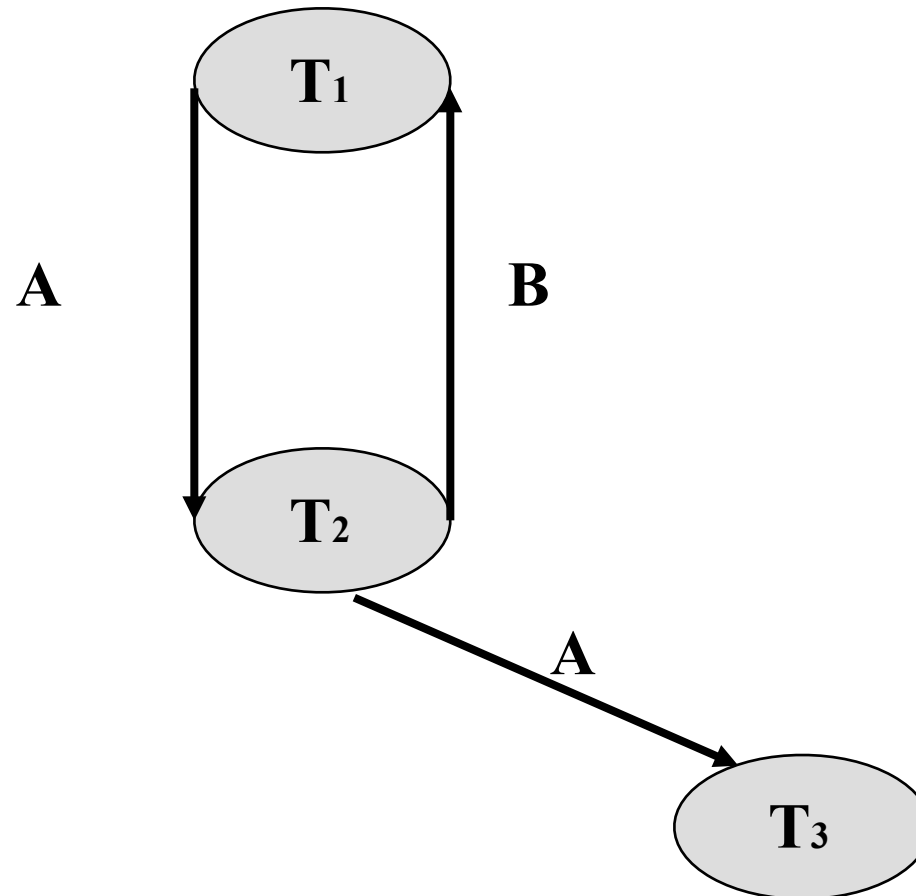
- On note  $T1 <_E T2$  s'il existe deux opérations conflictuelles  $o1$  et  $o2$  tel que  $o1$  apparaît avant  $o2$  dans E

- Graphe de précédence d'une exécution E

- Nœud : les transactions de E
- Arcs : la relation de précédence  $<_E$

# Exemple de graphe de précédence

**$r1(A)$   $w2(A)r2(B)$   $w2(B)$   $r3(A)w1(B)$**







## Test de sérialisabilité (2/2)

---

- Théorème

Une exécution est conflict-sérialisable **si et seulement si** son graphe de sérialisabilité est **acyclique**

- Tout ordre topologique des nœuds du graphe de précedence d'une exécution E constitue une exécution en série conflict-équivalente à E
  - Ordre topologique d'un graphe orienté acyclique : n'importe quel ordre total sur les sommet qui est 'compatible' avec le graphe (i.e., un ordre de visite des sommets tel qu'un sommet soit toujours visité avant ses successeurs)



# Sérialisabilité vs. conflict-sérializabilité

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



# Sérialisabilité vs. conflict-sérializabilité

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

# Sérialisabilité vs. conflict-sérializabilité

T1	T2	A	B
R(A, t)		25	25
t:=t+100			
W(A, t)			
	R(A, s)	125	
	<b>s:=s*1</b>		
	W(A, s)	<b>125</b>	
	R(B, s)		
	<b>s:=s*1</b>		
	W(B, s)		25
R(B, t)			
t:=t+100			
W(B, t)			<b>125</b>

**Non conflict-serializable**

**... But serializable**

# Sérialisabilité vs. conflict-sérializabilité

T1	T2	T3	A	B
R(A, t)			25	25
t:=t+100				
W(A, t)				
	R(A, s)		125	
	s:=s*2			
	W(A, s)			
	R(B, s)			
	s:=s*2			
	W(B, s)			50
R(B, t)				
t:=t+100				
W(B, t)		W(B,10)		150
		W(A,10)	10	10

**Non conflict-serializable**

**.. But serializable**



# Techniques de contrôle de concurrence (1/2)

---

## ■ Objectif

Coordonner l'exécution simultanée de transactions afin d'assurer la sérialisabilité

## ■ Techniques

### ■ Verrouillage

- Garantir la sérialisabilité en imposant des accès aux données en exclusion mutuelle

- Risque de verrou mortel (deadlock)

### ■ Estampillage

L'ordre d'exécution est fixé à priori suivant l'ordre d'apparition des transactions



## Techniques de contrôle de concurrence (2/2)

- Deux types d'approches
  - Optimiste
  - Pessimiste
- Algorithmes de contrôle de concurrence
  - Approche pessimiste
    - Verrouillage
    - Ordonnancement par estampillage
    - Hybride : verrouillage-estampillage
  - Approche optimiste
    - Verrouillage
    - Ordonnancement par estampillage



# Le verrouillage

---

## ■ Notion de verrou

- Variable décrivant l'état d'un élément d'une BD par rapport aux opérations possibles sur cet élément
- Permet la synchronisation des accès à un éléments de la BD par des transactions concurrentes

## ■ Granularité du verrou

- Choix de l'unité de verrouillage
  - Table
  - Page
  - Tuple

## ■ Un planificateur de verrous (lock scheduler)

A la réception d'une opération, trois options possibles :

- Planification immédiate de l'opération
- Différer l'opération
- Rejeter l'opération

**Décision en temps réel**

Règles simples pour garantir des  
**exécutions conflict-sérialisables**





# Le verrouillage (cont.)

---

- Opérations supplémentaires

- Verrouillage des données
- Déverrouillage des données

- Utilisation propre des verrous

- Consistance des transactions (transactions bien formées)
  - Une transaction doit d'abord verrouiller une données avant de la lire ou de la modifier
  - Si une transaction verrouille une donnée, elle doit la déverrouiller à un moment donné
- Légalité des plans
  - Deux transactions ne peuvent pas verrouiller des données en même temps



# Types de verrous

---

## ■ Verrou binaire

- Deux états (Locked et Unlocked)
- Deux opérations (Lock et Unlock)
- Assure l'exclusion mutuelle

## ■ Verrou partagé/exclusif

- Trois états
  - read\_locked (rl), write\_locked (wl) et unlocked (ul)
- Trois opérations
  - Read\_lock, Write\_lock et Unlock



# Algorithmes basés sur le verrouillage

- Les transactions doivent acquérir un verrou avant d'exécuter une opération sur une donnée
- Compatibilité des verrous

	<b>rl</b>	<b>wl</b>
<b>rl</b>	V	F
<b>wl</b>	F	F



# Notions de transaction bien formée et exécution légale

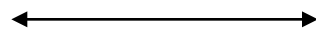
---

## Règle 1 : transaction bien-formée

$T_i: \dots li(A) \dots pi(A) \dots ui(A) \dots$

## Règle 2 : exécution légale

$T_i: \dots li(A) \dots pi(A) \dots ui(A) \dots$



pas de  $lj(A)$



## Exercice

---

Quelles sont les exécutions legales ?

Quelles sont les transactions bien-formées ?

$S1 = l1(A)l1(B)r1(A)w1(B)l2(B)u1(A)u1(B)r2(B)w2(B)u2(B)l3(B)r3(B)u3(B)$

$S2 = l1(A)r1(A)w1(B)u1(A)u1(B)l2(B)r2(B)w2(B)l3(B)r3(B)u3(B)$

$S3 = l1(A)r1(A)u1(A)l1(B)w1(B)u1(B)l2(B)r2(B)w2(B)u2(B)l3(B)r3(B)u3(B)$

# Sérialisabilité vs. conflict-sérializabilité

T1	T2	A	B
$I_1(A)$		25	25
$R(A, t)$			
$t := t + 100$			
$W(A, t)$			
$u_1(A)$			
	$I_2(A)$		
	$R(A, s)$		
	$s := s * 2$		
	$W(A, s)$		
	$u_2(A)$		
	$I_2(B)$		
	$W(B, s)$		
	$u_2(B)$		
$I_1(B)$			
$R(B, t)$			
$t := t + 100$			
$W(B, t)$			
$u_1(B)$			

Transactions bien-formées ?

Plan légal ?

Exécution sérialisable ?

250

.. Il faut également un protocole de verrouillage

50

150

# Protocole 2PL

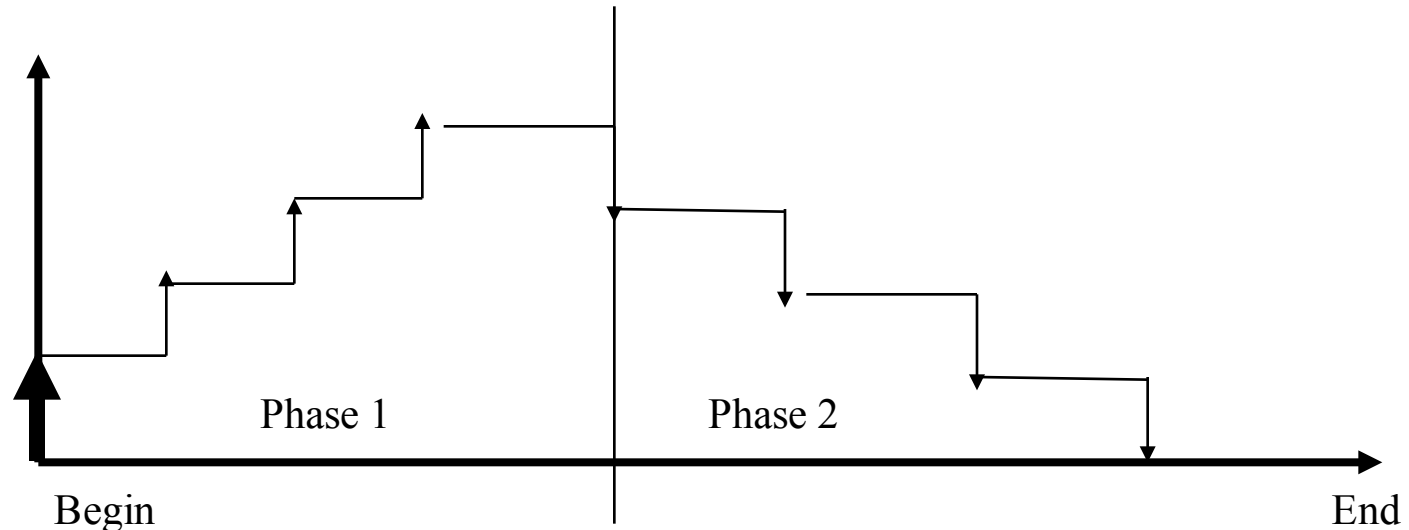
- Protocole de verrouillage en deux phases

Pour chaque transaction, toutes les opérations de verrouillage (lock) **précèdent** toutes les opérations de déverrouillage (unlock)

- Deux phases

- Phase d'acquisition des verrous
- Phase de relâchement des verrous

Règle 3 : les transactions doivent suivre le protocole 2PL





# Sérialisabilité vs. conflict-sérializabilité

---

T1

$I_1(A)$

$R(A, t)$

$t := t + 100$

$W(A, t)$

$u_1(A)$

$I_1(B)$

$R(B, t)$

$t := t + 100$

$W(B, t)$

$u_1(B)$

T2

$I_2(A)$

$R(A, s)$

$s := s * 2$

$W(A, s)$

$u_2(A)$

$I_2(B)$

$R(B, s)$

$s := s * 2$

$W(B, s)$

$u_2(B)$

Est-ce que ces transactions suivent  
le protocole 2PL ?



# Sérialisabilité vs. conflict-sérializabilité

T1	T2	A	B
$I_1(A)$		25	25
R(A, t)			
t:=t+100			
W(A, t)		125	
$u_1(A)$	$I_2(A)$		
	R(A, s)		
	s:=s*2		
	W(A, s)		
	$u_2(A)$	250	
	$I_2(B)$		
	R(B, s)		
	s:=s*2		
	W(B, s)		50
$I_1(B)$	$u_2(B)$		
R(B, t)			
t:=t+100			
W(B, t)			150
$u_1(B)$			



# Théorème

---

*Règles 1, 2 et 3  $\Rightarrow$  une exécution conflict-sérialisable  
 $\Rightarrow$  une exécution sérialisable*



## Au-delà de 2PL

---

- Améliorer les performances et augmenter le degré de concurrence
  - Verrous partagés
  - Différents niveaux de granularité
  - Autres techniques de gestion de la concurrence



# Verrous partagés

---

$$P = \dots l_1(A) \ r_1(A) \ u_1(A) \ \dots \ l_2(A) \ r_2(A) \ u_2(A)$$

P ne suit pas 2PL et pourtant il est sérialisable ...

Utilisation d'un verrou partagé:

$$P' = \dots \textcolor{red}{ls1(A)} \ r_1(A) \ \textcolor{red}{ls2(A)} \ r_2(A) \ \dots \ us1(A) \ us2(A)$$



# Verrous partagés vs. exclusif

Deux modes de verrouillages : partagé (**S**) ou exclusif (**X**)

## ■ Règle 1 : transactions bien-formées

$T_i = \dots l-S_1(A) \dots r_1(A) \dots u_1(A) \dots$

$T_i = \dots l-X_1(A) \dots w_1(A) \dots u_1(A) \dots$

*Mais si  $T_i$  lit et modifie le même objet ..*

Option 1: demande d'un verrou exclusif

$T_i = \dots l-X_1(A) \dots r_1(A) \dots w_1(A) \dots u(A) \dots$

Option 2: Mise à niveau du verrou

$T_i = \dots l-S_1(A) \dots r_1(A) \dots l-X_1(A) \dots w_1(A) \dots u(A) \dots$

# Verrous partagés vs. exclusif (cont.)

Deux modes de verrouillages : partagé (S) ou exclusif (X)

## ■ Règle 2 : exécutions légales

$S = \dots l-S_i(A) \dots \dots u_i(A) \dots$   
 $\longleftrightarrow$   
 pas de  $l-X_j(A)$

$S = \dots l-X_i(A) \dots \dots u_i(A) \dots$   
 $\longleftrightarrow$   
 pas de  $l-X_j(A)$   
 pas de  $l-S_j(A)$

*Matrice de compatibilité*

	S	X
S	Vrai	Faux
X	Faux	Faux



# Verrous partagés vs. exclusif (cont.)

Deux modes de verrouillages : partagé (S) ou exclusif (X)

## ■ Règle 3 : transactions 2PL

Pas de changement sauf dans le cas d'une mise à niveau

- Si la mise à niveau obtient plus de verrous  
(e.g.,  $S \rightarrow \{S, X\}$ )
  - pas de modification!
- (II) Si la mise à niveau relâche un verrou partagé  
(e.g.,  $S \rightarrow X$ )
  - autorisé dans la phase d'escalade

## Théorème

Règles 1, 2, 3 pour les verrous S/X  $\Rightarrow$  Exécutions  
sérialisables



# Verrouillage : avantages et inconvénients

- Un mécanisme de verrouillage permet à une transaction de se réserver l'usage exclusif d'une donnée aussi longtemps que c'est nécessaire
- Un inconvénient du verrouillage est son coût élevé lorsque les transactions font référence à de nombreuses données de la base
- Le degré de parallélisme est maximum puisque seules sont verrouillées les données réellement manipulées par les transactions
- On doit aussi tenir compte du problème des verrous mortels





# L'estampillage

---

## ■ Estampille

- Identification de façon unique d'une transaction
- Créée avant l'exécution de la transaction
- Les estampilles sont gérées par le SGBD et sont assignées typiquement dans l'ordre chronologique de soumission des transactions au système
- Peut être considérée comme étant la date de début de la transaction

## ■ Estampille d'un objet

- Correspond à l'estampille de la dernière transaction qui a accédé à cet objet
- Deux types d'estampille sur un objet :
  - L'estampille en écriture : correspond à l'estampille la plus élevée parmi celles des transactions ayant effectué avec succès une opération de lecture sur l'objet : la plus jeune de ces transactions.
  - L'estampille en lecture : correspond à l'estampille la plus élevée parmi celles des transactions ayant effectué avec succès une opération d'écriture sur l'objet : la plus jeune de ces transactions



# Ordonnancement par estampillage

## ■ Principe

Mettre en œuvre un protocole d'ordonnancement par estampilles qui assure que les opérations conflictuelles de lecture et d'écriture sont exécutées conformément à l'ordre des estampilles

```
// Contrôle d'ordonnancement des transactions
```

```
Ecrire ( $T_i$ ,  $O$ ) {
```

```
// la transaction  $T_i$  demande l'écriture de l'objet  $O$ ;
```

```
si  $ts(T_i) < W(O)$  ou  $ts(T_i) < R(O)$  alors abort ( $T_i$ )
```

```
sinon exécuter_écrire ( $T_i$ ,  $O$ ) } ;
```

```
Lire ( $T_i$ ,  $O$ ) {
```

```
// la transaction  $T_i$  demande la lecture de l'objet  $O$ ;
```

```
si  $ts(T_i) < W(O)$  alors abort ( $T_i$ )
```

```
sinon exécuter_lire( $T_i$ ,  $O$ ) } ;
```



## Avantages et inconvénients de l'estampillage

---

- Pas de problème de blocage car aucune transaction n'attend une autre transaction
- Deux inconvénients majeurs
  - Livelock (reprises multiples d'une transaction)
  - Annulation en cascade



# Gestion des transactions en pratique ..

## ■ Verrouillage Implicite vs Verrouillage explicite

## ■ Niveaux de consistance

La définition des niveaux est basée sur la notion de données impropres et des possibilités (ou pas) pour une transaction T d'effectuer les opérations suivantes :

- a- T ne peut pas modifier les données impropres des autres transactions
- b- T ne peut valider aucune modification avant de valider toutes (i.e., jusqu'à la fin de la transaction)
- c- T ne peut pas lire les données impropres des autres transactions
- d- Les autres transactions ne peuvent pas rendre impropres des données lues par T tant que T ne s'est pas terminée

## ■ 4 niveaux possibles

- Niveau 3 : a-b-c-d
- Niveau 2 : a-b-c
- Niveau 1 : a-b
- Niveau 0 : a

- ⇒ Ces niveaux permettent plus de flexibilité dans la définition des transactions
- ⇒ Lien consistance-isolation : plus on monte dans la hiérarchie des niveaux, plus on augmente l'isolation



# Niveaux d'isolation - SQL 92

- Basés sur la notion de phénomènes (i.e., problèmes possibles si l'isolation n'est pas assurée)
  - Lectures impropres (**dirty read**)  
lecture de données impropres (**nonrepeatable read**)
  - Lectures non répétitives (**phantom read**)  
dans une même transaction, deux lectures d'une même donnée renvoient des résultats différents
  - Lignes fantômes
- 4 niveaux d'isolation emboîtés :

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
<i>Read uncommitted</i>	Possible	Possible	Possible
<i>Read committed</i>	Not Possible	Possible	Possible
<i>Repeatable read</i>	Not Possible	Not Possible	Possible
<i>Serializable</i>	Not Possible	Not Possible	Not Possible



# Transactions dans le SGBD Oracle (1/4)

---

## Niveaux d'isolation

- Read committed (default isolation level)
  - Uniquement les données propres (validées) sont visibles
  - Lectures non répétitives (**phantom read**)
- Serializable
- Read only (not part of SQL92)

SELECT last\_name, salary FROM employees  
WHERE last\_name IN ('Banda', 'Greene', 'Hintz');

UPDATE employees SET salary = 7000 WHERE  
last\_name = 'Banda';

INSERT INTO employees (employee\_id, last\_name, email,  
hire\_date, job\_id) VALUES (210, 'Hintz', 'JHINTZ',  
SYSDATE, 'SH\_CLERK');

LAST_NAME	SALARY
-----	-----
Banda	6200
Greene	9500

SET TRANSACTION ISOLATION LEVEL READ  
COMMITTED;

SELECT last\_name, salary FROM employees WHERE  
last\_name IN ('Banda', 'Greene', 'Hintz');

UPDATE employees SET salary = 9900 WHERE  
last\_name='Greene';

SELECT last\_name, salary FROM employees WHERE  
last\_name IN ('Banda', 'Greene', 'Hintz');

UPDATE employees SET salary = 6300 WHERE  
last\_name = 'Banda';

COMMIT;

SELECT last\_name, salary FROM employees WHERE  
last\_name IN ('Banda', 'Greene', 'Hintz');

COMMIT;

SELECT last\_name, salary FROM employees WHERE  
last\_name IN ('Banda', 'Greene', 'Hintz');



# Transactions dans le SGBD Oracle (1/4)

## Ordres liés à la gestion des transactions

- `Begin transaction`
- `Savepoint`
- `Commit`
- `Rollback`
- `Rollback to savepoint`
- `Set transaction`
- `BEGIN_DISCRETE_TRANSACTION` (validation différée)





## Transactions dans le SGBD Oracle (2/4)

- Deux niveaux de lectures consistantes
  - Statement-Level Read Consistency
    - Par défaut, lectures consistantes pour les ordres SQL : SELECT, INSERT avec une sous-requête, UPDATE et DELETE
  - Transaction-Level Read Consistency
    - Niveau d'isolation adéquat

**Set transaction level serializable**

**ALTER SESSION SET ISOLATION\_LEVEL  
SERIALIZABLE;**



# Transactions dans le SGBD Oracle (3/4)

---

## ■ Types de verrous

- Row Locks (TX)
  - Row Share (RS)
  - Row Exclusive (RX)
- Table Locks (TM)
  - Share Table (S)
  - Share Row Exclusive Table Locks (SRX)
  - Exclusive Table (X)
- Exemples

LOCK TABLE table IN SHARE MODE;

LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

LOCK TABLE table IN ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;

# Transactions dans le SGBD Oracle (4/4)

**Table 20-3 Summary of Table Locks**

SQL Statement	Mode of Table Lock	Lock Modes Permitted?				
		RS	RX	S	SRX	X
SELECT...FROM table...	none	Y	Y	Y	Y	Y
INSERT INTO table ...	RX	Y	Y	N	N	N
UPDATE table ...	RX	Y*	Y*	N	N	N
DELETE FROM table ...	RX	Y*	Y*	N	N	N
SELECT ... FROM table FOR UPDATE OF ...	RS	Y*	Y*	Y*	Y*	N
LOCK TABLE table IN ROW SHARE MODE	RS	Y	Y	Y	Y	N
LOCK TABLE table IN ROW EXCLUSIVE MODE	RX	Y	Y	N	N	N
LOCK TABLE table IN SHARE MODE	S	Y	N	Y	N	N
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE	SRX	Y	N	N	N	N
LOCK TABLE table IN EXCLUSIVE MODE	X	N	N	N	N	N
RS: row share RX: row exclusive S: share SRX: share row exclusive X: exclusive		*Yes, if no conflicting row locks are held by another transaction. Otherwise, waits occur.				