



Université Constantine 2
جامعة قسنطينة 2

Développement Avancé d'Applications Web

– Chapitre 4 –

Transactions & Contrôle de la Concurrency

Dr Bouanaka Chafia

NTIC

chafia.bouanaka@univ-constantine2.dz

Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
Nouvelles technologies	TLSI	Licence 3	Génie Logiciel (GL)

Objectif du cours

- Comprendre la problématique des **transactions** en BD
- Comprendre la problématique **des accès concurrents** en BD
- Comprendre **les classes de solutions** aux problèmes d'accès concurrents

Plan du Cours

- Notion de transaction dans les BDs
 - Définitions
 - Propriétés ACID d'une transaction
- Problèmes de l'accès concurrent à une BD
- Gestion de la concurrence
 - Ordonnancement des transactions concurrentes
 - Capacité de c-sérialisation
 - Contrôle d'accès à une BD
 - Test de c-sérialisation
 - Les verrous

Section 1:

Notion de Transaction dans les BDs

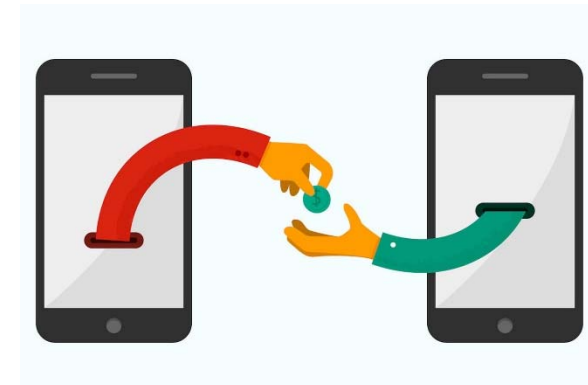
Notion de transaction

Exemple introductif : Transaction bancaire

- Transaction bancaire :
 - Virement Bancaire d'un montant de 10000 DA
 - Compte débité : A
 - Compte crédité : B

Pour réaliser cette transaction,

- Deux opérations doivent être exécutées :
 - Retirer 10000 DA du compte A
 - Ajouter 10000 DA au compte B



Avant la transaction		
NumCompte	Client	Solde
A	X	100000
B	Y	75000
...

Après la transaction		
NumCompte	Client	Solde
A	X	90000
B	Y	85000
...

Notion de transaction

Exemple introductif : Transaction bancaire

- Au niveau implémentation, ce virement est réalisé par l'exécution des opérations suivantes sur les comptes **A**(Client X) et **B**(Client Y) :
 - **Read**(A.solde)
 - $A.solde = A.solde - 10000$
 - **Write**(A.solde)

 - **Read**(B.solde)
 - $B.solde = B.solde + 10000$
 - **Write**(B.solde)

Au niveau SQL, on aura :

Débit(A)

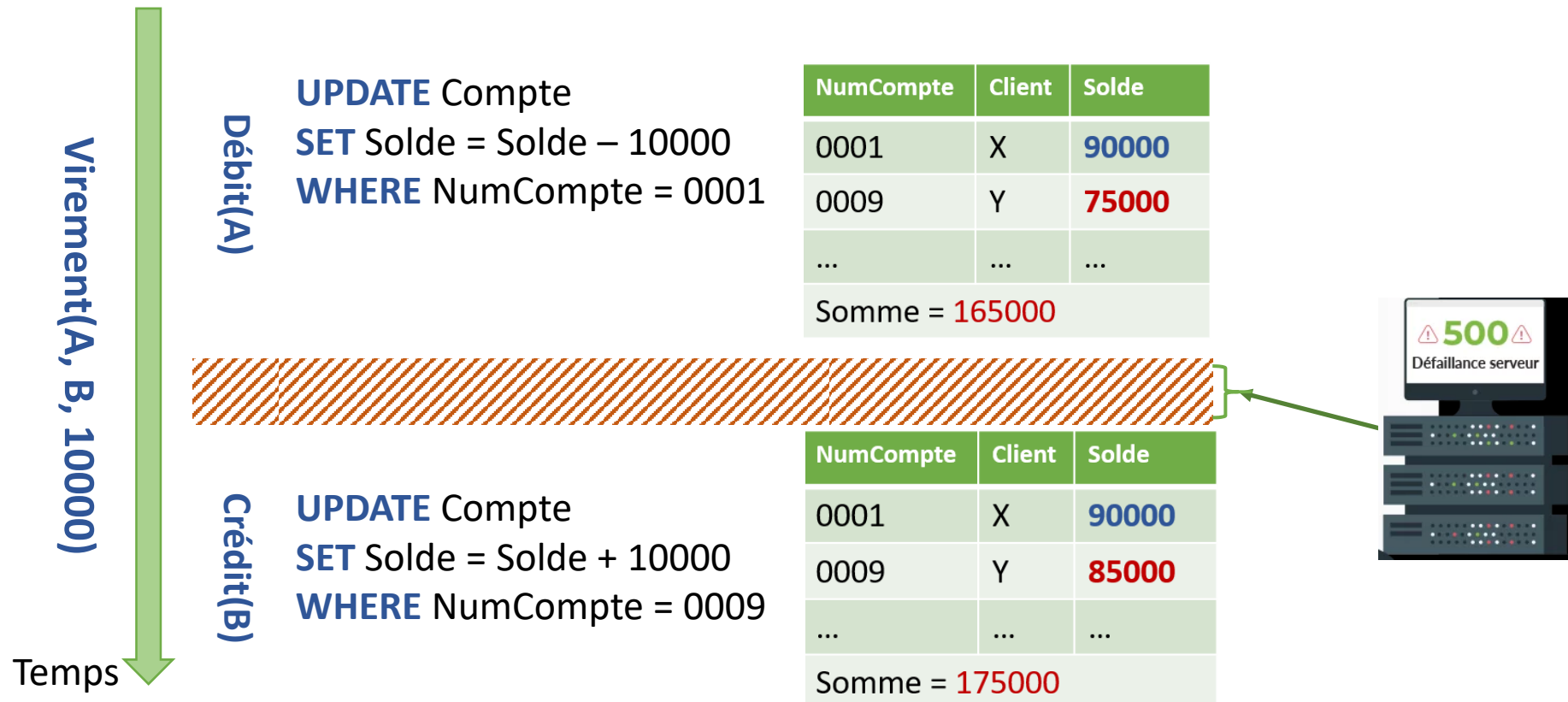
```
UPDATE Compte  
SET Solde = Solde - 10000  
WHERE NumCompte = 0001
```

Crédit(B)

```
UPDATE Compte  
SET Solde = Solde + 10000  
WHERE NumCompte = 0009
```

Notion de transaction

Contexte Mono-utilisateur



Notion de transaction

Contexte Mono-utilisateur

Différents problèmes peuvent apparaître avant la fin du virement bancaire :

- L'arrêt du serveur de bases de données;
- Une erreur de programmation entraînant l'arrêt de l'application;
- La violation d'une contrainte amenant le système à rejeter les opérations demandées;
- Une annulation décidée par l'utilisateur...

Afin d'assurer la cohérence de la BD, les opérations de **crédit/débit** doivent **s'exécuter toutes les deux** ou bien **annuler** toutes les deux.



Notion de transaction



Atomicité de la transaction

Notion de transaction

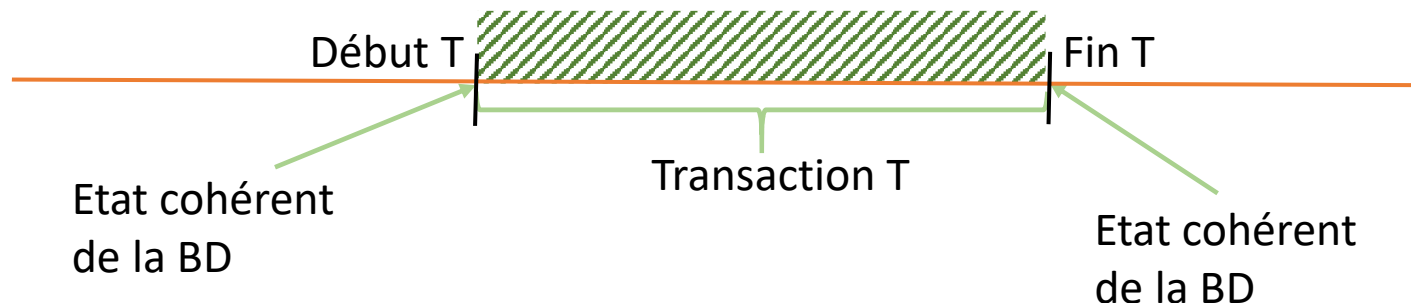
Transaction

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.

Les transactions accèdent aux données en utilisant les opérations **Read/Write**

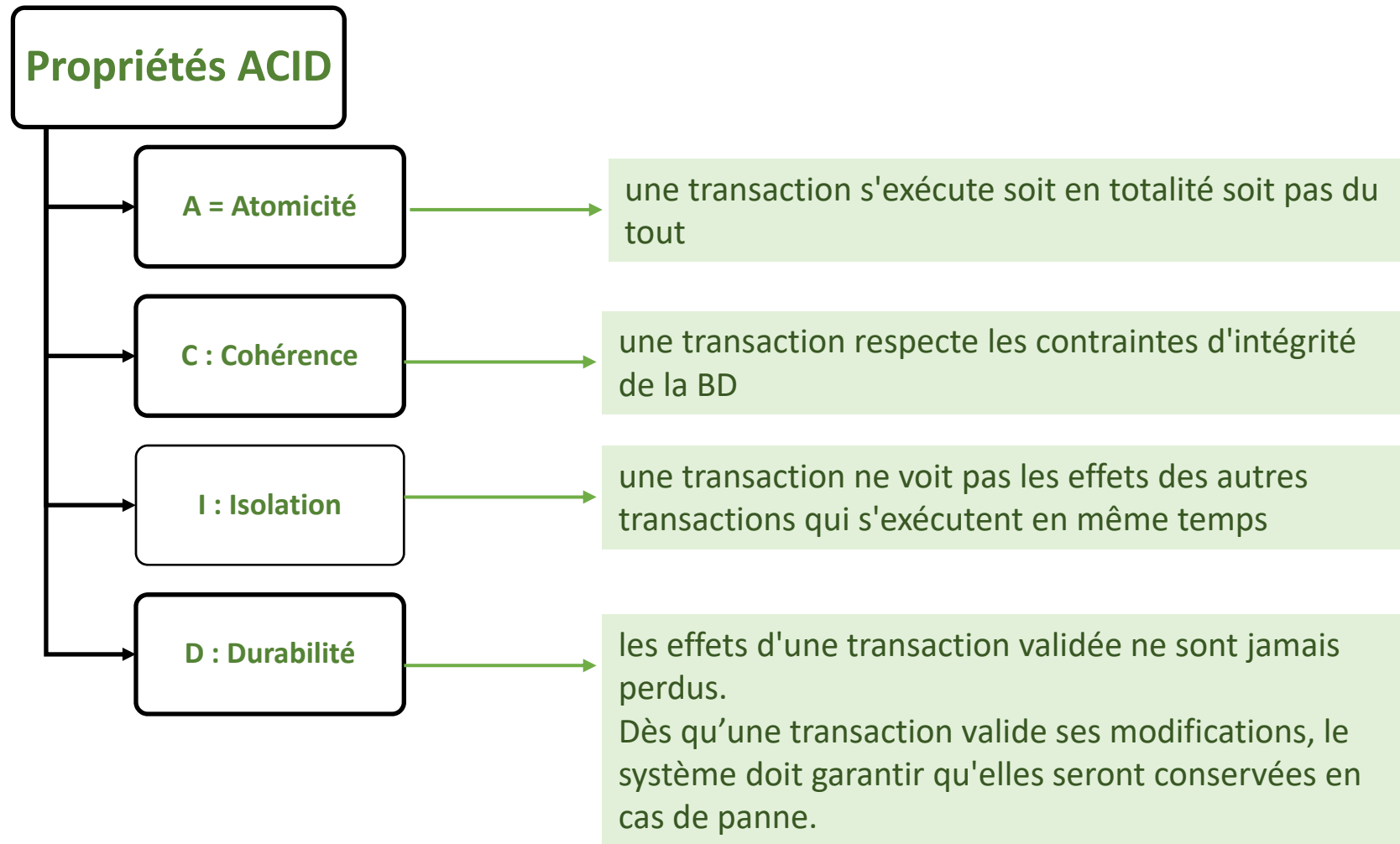
Définition : Transaction

- On appelle **transaction** un ensemble séquentiel d'opérations d'un seul utilisateur permettant de passer d'un **état cohérent** de la BD à **un autre état cohérent**.
- L'ensemble des opérations d'une transaction est **atomique**
- **Une transaction est une unité logique de travail sur la base de données.**



Notion de transaction

Propriétés ACID d'une transaction



Notion de transaction

Propriétés ACID : Atomicité

- Une transaction doit effectuer toutes ses opérations ou ne rien faire du tout.
- En cas d'échec, le système doit annuler toutes les modifications que la transaction a effectuée.
- Une exécution partielle de la transaction est inacceptable :
 - Une transaction qui ne peut pas se terminer doit être annulée
 - Le SGBD doit annuler toutes les modifications (écritures)
- L'atomicité est menacée par :
 - les **pannes de programme** : l'annulation du débit si solde insuffisant
 - du **système** : blocage ou incohérence dans l'exécution concurrente
 - ou du **matériel**.

Notion de transaction

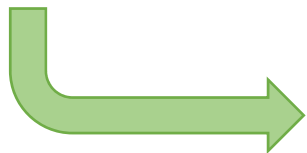
Propriétés ACID : Atomicité

Pour assurer l'atomicité, on utilise les primitives :

- **Commit** : permet de **valider** tous les changements sur la BD, si toutes les opérations d'une transaction sont exécutées avec succès,
- **Rollback** : permet d'**annuler** tous les changements sur la BD, si l'une des opérations de la transaction échoue.

Important

- Après un **Commit** (validation d'une transaction), les changements effectués sur la BD ne peuvent plus être annulés.



Durabilité des changements

Notion de transaction

Propriétés ACID : Atomicité

Pour l'exemple du virement bancaire :

- Atomicité : Exécuter **Débit(A)** et **Crédit(B)** comme une seule unité

Avant : A = 100000	B = 75000
Transaction T	
Débit(A)	Crédit(B)
Read (A.solde) A.solde = A.solde - 10000 Write (A.solde)	Read (B.solde) B.solde = B.solde + 10000 Write (B.solde)
Après : A = 90000	B = 85000

Notion de transaction

Propriétés ACID : Cohérence

- Une transaction mène la BD d'un **état cohérent** vers **un autre état cohérent**
- Pendant l'exécution de la transaction l'état peut être incohérent.

Pour assurer la cohérence de la BD:

- ☐ une transaction respecte les contraintes d'intégrité de la BD :
 - les contraintes de clé primaire (clause primary key);
 - l'intégrité référentielle (clause foreign key);
 - les contraintes check;
 - les contraintes implantées par triggers.
- ☐ une transaction respecte les contraintes exprimant la cohérence de la BD
 - Contraintes sur les données: $\text{compte.solde} \geq 0$, en permanence
 - Contraintes sur les opérations: dans un transfert entre deux comptes

$$\text{source.solde} + \text{dest.solde} = \text{constant}$$

Notion de transaction

Propriétés ACID : Cohérence

Pour l'exemple de la transaction bancaire :

Les contraintes d'intégrité sont assurés par :


$$\mathbf{A.solde + B.solde = 17500}$$

Total Avant la transaction	Total Après la transaction
Transaction T	
Débit(A)	Crédit(B)
Read (A.solde) A.solde = A.solde - 10000 Write (A.solde)	Read (B.solde) B.solde = B.solde + 10000 Write (B.solde)
Avant T : 100000 + 75000	Après T : 90000 + 85000
Total = 17500	

Notion de transaction

Structure et Etats d'une Transaction

Une transaction est constituée de trois primitives :



Etape	Primitive
Ouverture d'une transaction	Begin transaction
Travail sur les données	Commandes SQL ou PL/SQL
Clôture avec confirmation Clôture avec annulation	COMMIT ROLLBACK

Une transaction peut être dans l'un des trois états suivants :

- **Exécution Normale**
 - La transaction arrive sur la fin
 - Point de confirmation : **commit**
- **Un assassinat**
 - Arrêt par un événement extérieur
 - Arrêt par le SGBD lui même (deadlock)
 - Le système fait marche arrière (**rollback**) et annule les actions déjà effectuées
- **Un suicide**
 - Arrêt et annulation par la transaction elle même (**rollback**)

Section 2:

Problèmes de l'accès Concurrent à une BD

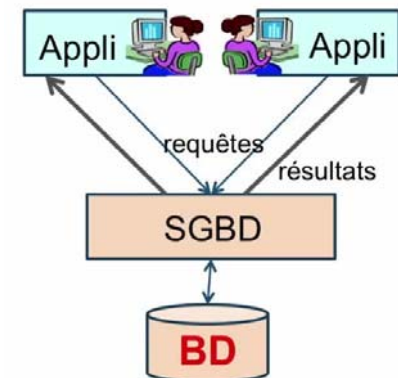
Problèmes d'accès concurrents

Contexte multi-utilisateurs

Concurrence des transactions :

Dans un système multi-utilisateurs : de nombreuses transactions s'exécutent en parallèle

- Vision client-serveur
 - Les transactions/programmes = clients qui envoient des demandes au SGBD (demandes de lecture ou d'écriture dans la BD)
 - Le SGBD = serveur qui reçoit ces demandes et les exécute
- Le SGBD exécute les opérations en séquence, pas en parallèle
 - Concurrence = **entrelacement** des opérations des différentes transactions
- Besoin pour le SGBD d'être capable de gérer les transactions concurrentes.



Les primitives **Commit/rollback** sont alors **insuffisantes**

Problèmes d'accès concurrents

Contexte multi-utilisateurs


Quatre types de problèmes peuvent survenir lors de l'accès concurrents à une BD :

- Perte de mise à jour
- Lecture impropre
- Lecture non reproductible
- Objets fantômes

Problèmes d'accès concurrents

Problème : Perte de MAJs

T1 et T2 modifient simultanément **A**




Temps

T1	T2	BD
		A = 10
read A		
	read A	
A = A + 10		
write A		A = 20
	A = A + 50	
	write A	A = 60
	commit	
commit		

Les modifications effectuées par **T1** sont perdues

Problèmes d'accès concurrents

Problème : Lecture impropre



Temps

T1	T2	BD
		A + B = 200
		A = 120 B = 80
read A		
A = A - 50		
write A		A = 70
	read A	
	read B	
Rollback		
	Display(A + B) (150 est affiché)	

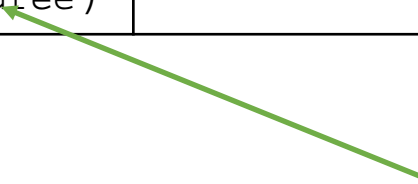
La transaction **T1** échoue et annule les modifications sur **A**

T2 lit une valeur de **A** non validée, affiche une valeur incohérente

Problèmes d'accès concurrents

Problème : Lecture impropre

T1	T2	BD
		A = 50
	A = 70	
	write A	A = 70
read A (70 est lu)		
	Rollback (La valeur initiale de A est restaurée)	A = 50



La transaction **T2** échoue et annule les modifications sur **A**

T1 lit une valeur de **A** non confirmée et utilise une valeur incohérente

Problèmes d'accès concurrents

Problème : objets fantômes

T1	T2	BD
		{1, 2, 3}
Afficher card(E) 3 est affiché		
	Insert 4 into E	{1, 2, 3, 4}
Afficher card(E) 4 est affiché		

L'objet **4** est fantôme pour **T1**



Section 3:

Gestion de la concurrence

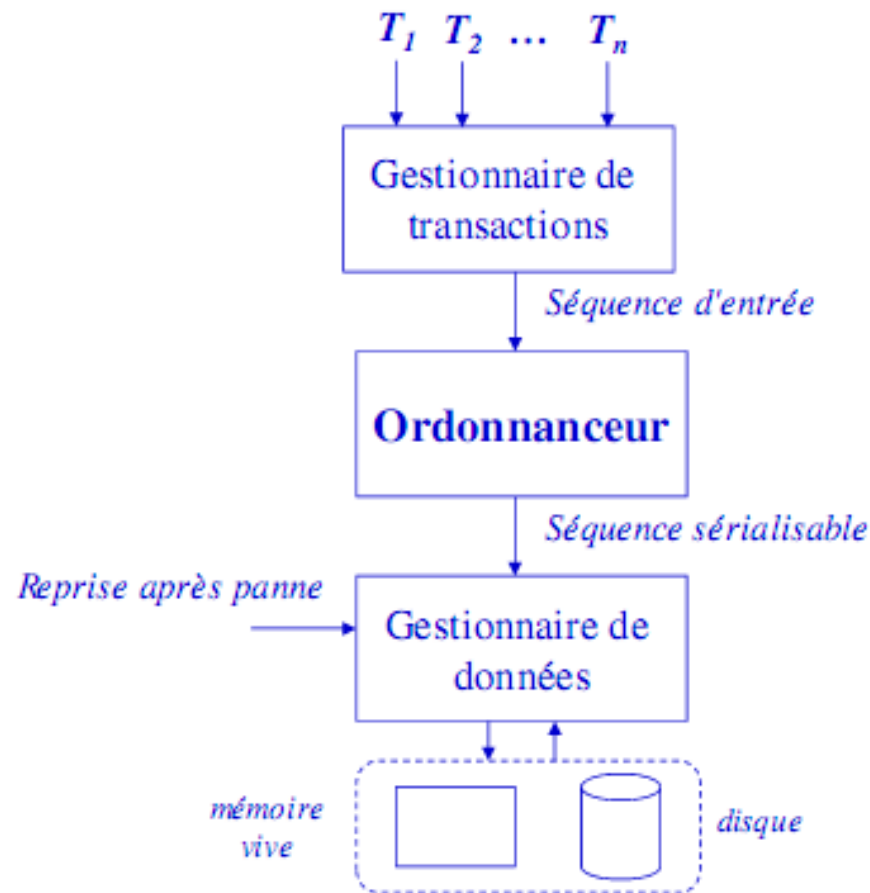
Gestion de la concurrence

Objectifs de la gestion de la concurrence

- La concurrence permet l'exécution de plusieurs transactions simultanément pour :
 - ✓ Une meilleure utilisation du processeur (une transaction peut utiliser le processeur pendant qu'une autre accède au disque)
 - ✓ Une réduction du temps de réponse aux transactions (une transaction courte n'attend pas la fin d'une transaction longue)
- Le SGBD assure la transparence du partage des données entre les transactions en réalisant les tâches suivantes :
 - ✓ **entrelacer les actions des transactions** : **Ordonnancement** des opérations des différentes transactions,
 - ✓ **contrôler l'accès concurrent** aux mêmes ressources (une table, un tuple, un attribut) de la BD afin d'éviter les incohérences et d'assurer **l'isolation**.

Gestion de la concurrence

Composants SGBD pour la gestion des transactions



Gestion de la concurrence

Ordonnancement des transactions

Définition : Ordonnancement

Un Ordonnancement est une séquence chronologique spécifiant l'ordre d'exécution des opérations de plusieurs transactions.

Exemple d'ordonnancement :

Soit deux transactions T1 et T2 constituées des opérations suivantes :

- T1: (T1; lire; A); (T1; écrire; A)
- T2: (T2; lire; A); (T2; lire; B); (T2; écrire; A); (T2; écrire; B)

Un ordonnancement possible de T1 et T2 est :

- O : (T1; lire; A); (T2; lire; A); (T1; écrire; A); (T2; lire; B); (T2; écrire; A); (T2; écrire; B)

Définition : Ordonnancement Sériel

C'est un ordonnancement où les opérations de chaque transaction sont exécutées de manière **consécutif**, sans aucun entrelacement avec les opérations d'autres transactions.

Gestion de la concurrence

Ordonnancement des transactions

Définition : Ordonnancement non sériel

C'est un ordonnancement où les opérations d'un ensemble de transactions sont exécutées de manière entrelacée.

Définition : Ordonnancement sérialisable

Soit O un ordonnancement composé des transactions $\{T_1, \dots, T_n\}$

- O est **sérialisable** si son exécution donne le même résultat que l'exécution **sérielle** des transactions T_1, \dots, T_n dans un ordre quelconque.

Gestion de la concurrence

Capacité de sérialisation

Important

- Un ordonnancement **non sériel** est **sérialisable** s'il est équivalent à un ordonnancement **sériel**.

Trois types d'équivalences :

- de résultat : appelé **r-équivalence**
- de visibilité : appelé **v-équivalence**
- de conflit : appelé **c-équivalence**

Gestion de la concurrence

Capacité de sérialisation

Définition : r-équivalence

Deux ordonnancements des transactions T_1, \dots, T_n sont équivalentes relativement aux résultats ssi :

- leur exécution détermine **le même état de la BD** :
 - pour un état initial de la BD
 - pour tout état initial de la BD

Remarque

- Deux états de BD sont égaux si tout élément d'un état a la même valeur que son correspondant dans l'autre état

Gestion de la concurrence

Capacité de sérialisation

Définition : v-équivalence

Deux ordonnancements O_1, O_2 des transactions T_1, \dots, T_n sont équivalents relativement aux vues ssi :

- l'ordre relatif de deux opérations de **lecture/écriture** sur **un objet** de la BD est toujours le même dans les deux ordonnancements.

Définition : c-équivalence

Deux ordonnancements O_1, O_2 des transactions T_1, \dots, T_n sont équivalents relativement aux conflits ssi :

- l'ordre relatif de deux opérations **conflictuelles** est toujours le même dans les deux ordonnancements.

Gestion de la concurrence

Capacité de sérialisation

Opérations commutables / conflictuelles

- Deux opérations A et B sont **commutables** ssi l'exécution A;B (A suivi de B) donne le même résultat que B;A :
 - Les valeurs finales des objets manipulés sont les mêmes
 - Les valeurs obtenues par lecture sont les mêmes
- Opérations conflictuelles: qui ne sont pas commutables

Définition :

Les opérations t_i et t_j des transactions T_i et T_j sont en conflit s'il existe un objet Q accédé par t_i et t_j et l'une d'elles écrit Q .

$t_i \backslash t_j$	Lire(Q)	Ecrire(Q)
Lire (Q)	Pas de conflit	en Conflit
Ecrire(Q)	en Conflit	en Conflit

Gestion de la concurrence

Capacité de sérialisation

Parmi tous les ordonnancements possibles d'un ensemble de transactions, il est nécessaire d'identifier les **ordonnements c-équivalents**.

Exemples d'ordonnements équivalents :

Soient deux transactions T1, T2 manipulant toutes les deux les ressources A et B

T_1	T_2
<i>Lire(A)</i> $A := A - 1000$ <i>Ecrire(A)</i> <i>Lire(B)</i> $B := B + 1000$ <i>Ecrire(B)</i>	<i>Lire(A)</i> $Temp := A * 0,1$ $A := A - Temp$ <i>Ecrire(A)</i> <i>Lire(B)</i> $B := B + Temp$ <i>Ecrire(B)</i>

Ordonnement O1
(exécution **en série** de T1 et T2)

T_1	T_2
<i>Lire(A)</i> $A := A - 1000$ <i>Ecrire(A)</i>	<i>Lire(A)</i> $Temp := A * 0,1$ $A := A - Temp$ <i>Ecrire(A)</i>
<i>Lire(B)</i> $B := B + 1000$ <i>Ecrire(B)</i>	<i>Lire(B)</i> $B := B + Temp$ <i>Ecrire(B)</i>

Ordonnement O2
(exécution **entrelacé** de T1 et T2)

T_1	T_2
<i>Lire(A)</i> $A := A - 1000$	<i>Lire(A)</i> $Temp := A * 0,1$ $A := A - Temp$ <i>Ecrire(A)</i> <i>Lire(B)</i>
<i>Ecrire(A)</i> <i>Lire(B)</i> $B := B + 1000$ <i>Ecrire(B)</i>	$B := B + Temp$ <i>Ecrire(B)</i>

Ordonnement O3
(exécution **entrelacé** de T1 et T2)



- O1 et O2 sont **c-équivalents**,
- O1 et O3 ne sont pas c-équivalents puisque l'ordre des opérations conflictuelles n'est pas respecté

Contrôle de la concurrence

Test de c-sérialisation : Graphe de précédence

Définition :

Soit un ordonnancement O des transactions $\{T_1, \dots, T_n\}$.

Le graphe de précédence de O est un graphe (N, A) , avec N est l'ensemble des nœuds et A est l'ensemble des arcs du graphe; tel que :

- N est l'ensemble des transactions
- Il y a un arc (T_i, T_j) s'il y a un conflit entre T_i et T_j sur un objet Q et T_i accède à Q avant T_j

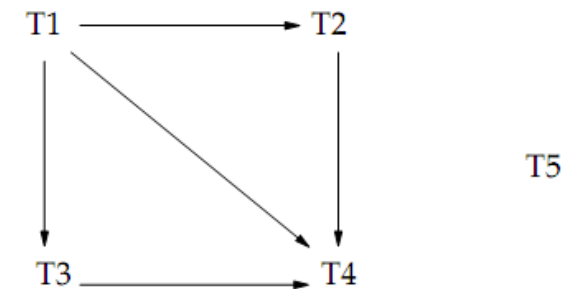


O est **c-sérialisable** ssi son graphe de précédence est **acyclique** (ne contient pas de cycles).

Contrôle de la concurrence

Test de c-sérialisation : Construire le Graphe de précédence

temps	T1	T2	T3	T4	T5
t1		Lire(x)			
t2	Lire(y)				
t3	Lire(Z)				
t4					Lire(V)
t5					Lire(W)
t6					Ecrire(W)
t7		Lire(y)			
t8		Ecrire(Y)			
t9			Ecrire(Z)		
t10	Lire(U)				
t11				Lire(Y)	
t12				Ecrire(Y)	
t13				Lire(Z)	
t14				Ecrire(Z)	
t15	Lire(U)				
t16	Ecrire(U)				



Graphe de précédence

Les transactions T1, T2, T3, T4, T5 sont c-sérialisables

Contrôle de la concurrence

Test de c-sérialisation : Construire le Graphe de précédence

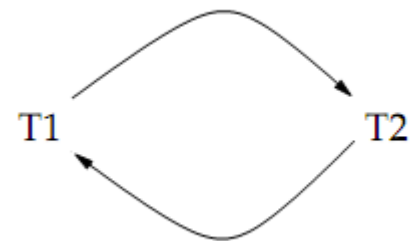
Exemple 1

Soit l'ordonnancement suivant :

temps	T1	T2
t1	Lire(x)	
t2		Lire(y)
t3	Ecrire(y)	
t4	commit	
t5		Ecrire(y)
t6		commit

Diagram illustrating conflicts between transactions T1 and T2:

- Conflict between T1's `Ecrire(y)` at t3 and T2's `Lire(y)` at t2.
- Conflict between T1's `commit` at t4 and T2's `Ecrire(y)` at t5.



Graphe de précédence

Graphe de précédence cyclique → Ordonnancement non sérialisable

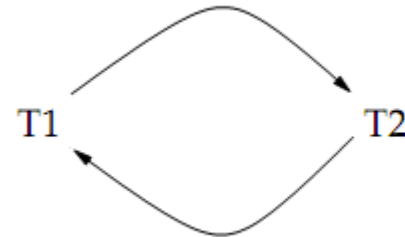
Les transactions T1, T2 ne sont pas c-sérialisables

Contrôle de la concurrence

Test de c-sérialisation : Construire le Graphe de précédence

Exemple 2

temps	T1	T2
t1	Begin transaction	
t2	Lire(solde_x)	
t3	Solde_x = solde_x +100	
t4	Ecrire(solde_x)	
t5		Begin_transaction
t6		Lire(solde_x)
t7		Solde_x = solde_x *1,1
t8		Ecrire(solde_x)
t9		Lire(solde_y)
t10		Solde_y = solde_y *1,1
t11		Ecrire(solde_y)
t12		commit
t13	Lire(solde_y)	
t13	Solde_y = solde_y- 100	
t14	Ecrire(solde_y)	
t15	commit	



Graphe de précédence cyclique



Ordonnancement non-sérialisable

- Initialement : solde_x = 100 et solde_y = 400.
- L'exécution serielle de T1 & T2 donne :
–T1; T2 : **solde_x = 220** et **solde_y = 330**.
–T2 ; T1 : **solde_x = 210** et **solde_y = 340**.
- Résultat de l'ordonnancement précédant est :
solde_x = 220 et **solde_y = 340**.
- Résultat incorrecte : n'est obtenu par aucune exécution sérielle

Contrôle de la concurrence

Les verrous

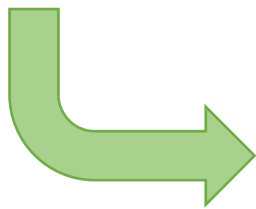
Cause des problèmes de l'accès concurrent :

La principale cause des problèmes d'accès à une BD est l'**Ordre d'exécution** des **Actions primitives (Read, Write)** sur des données partagées dans des transactions différentes.

Constat :

Tester si un ordonnancement est c-sérialisable après son exécution (ou bien vérifier sur son graphe de précédence) est inefficace.

La solution aux quatre problèmes rencontrés lors d'un accès concurrent à une BD



Isolation des transactions

Contrôle de la concurrence

Les Verrous

L'isolation : une transaction ne voit pas les effets des autres transactions qui s'exécutent en même temps (**Concurrence**)

- Les écritures des autres transactions en cours sur les données manipulés par la transaction en question ne sont pas visibles
- C'est comme si chaque transaction s'exécutait toute seule dans le SGBD

Comment assurer l'isolation :

- Définir des stratégies de contrôle de concurrence qui puissent garantir la c-sérialisation.
- Ces protocoles n'auront pas besoin de faire le test sur le graphe de précedence



Le verrouillage des ressources partagées

Contrôle de la concurrence

Les verrous

Définition : Verrouillage

Une stratégie employée pour contrôler les accès concurrents aux données. Lorsqu'une transaction accède à une donnée, **un verrou** est utilisé pour bloquer d'autres transactions à accéder à cette donnée pour éviter de **faux résultats**.

Deux types de verrous sont autorisés sur les données d'une BD :

Définition : Verrou partagé(Shared)

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

Définition : Verrou eXclusif

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

Contrôle de la concurrence

Les verrous : Algorithme de Verrouillage

Toute transaction devant accéder à une donnée verrouille d'abord la donnée, en demandant:

- soit un verrouillage partagé (**Shared**)
- 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 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'exécution
 - implicitement lorsqu'elle se termine (par une annulation ou une validation).

Remarque

- Les effets d'une opération d'écriture ne seront visibles aux autres transactions que lorsque le verrou exclusif sur cette donnée est libéré.

Contrôle de la concurrence

Les verrous : Algorithme de Verrouillage

Phase 1 : Verrouillage

Lecture : Verrou « S » (Shared)

Ecriture : Verrou « X » (Exclusive)

Tenter d'exécuter une lecture (SELECT) ou une écriture (UPDATE)
= essayer de poser un verrou

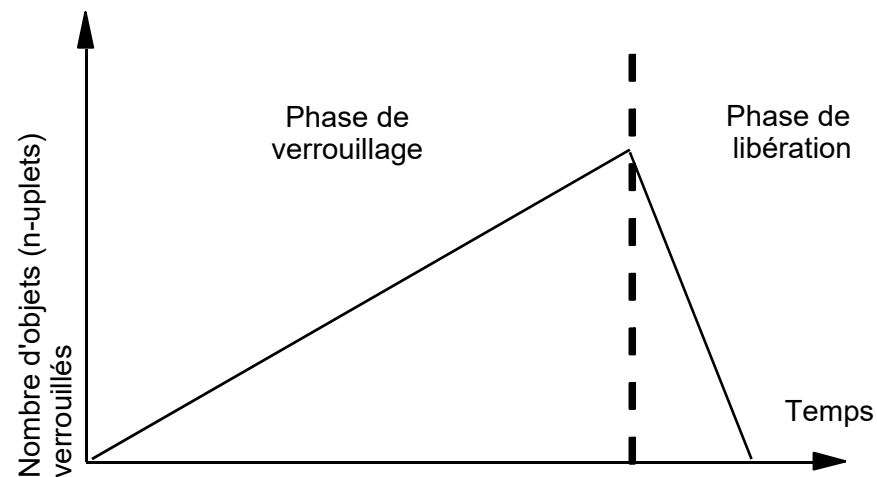
Verrou demandé \ Verrou détenu	S	X
	S	X
S	Accordé	Mise en attente
X	Mise en attente	Mise en attente

Contrôle de la concurrence

Les verrous : Algorithme de Verrouillage

Phase 2 : libération

Fin d'une transaction = retirer les verrous et libérer toutes les ressources



Contrôle de la concurrence

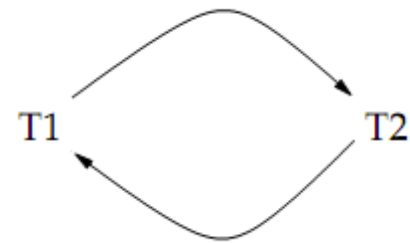
Les verrous : Exemple1

Pour l'exemple de l'ordonnancement :

temps	T1	T2
t1	Lire(x)	
t2		Lire(y)
t3	Ecrire(y)	
t4	commit	
t5		Ecrire(y)
t6		commit

Diagram illustrating conflicts between T1 and T2 operations on variable y:

- Conflict between T1's `Ecrire(y)` at t3 and T2's `Lire(y)` at t2.
- Conflict between T1's `Ecrire(y)` at t3 and T2's `Ecrire(y)` at t5.



Graphe de précedence

Graphe de précedence cyclique → Ordonnancement non sérialisable

L'utilisation des verrous resoud le problème

Contrôle de la concurrence

Les verrous : Exemple1

- **Utilisation de verrous pour l'exemple de la diapo 44:**

- $r1(x)$: acceptée, un sous-verrou de lecture déposé sur x et donné à $T1$
- $r2(y)$: acceptée, sous-verrou de lecture déposé sur y et donné à $T2$
- $w1(y)$: bloquée (conflit avec $r2(y)$),

le sous-verrou de lecture sur y donné à $T2$ empêche de donner le sous-verrou d'écriture sur y à $T1$

- $c1$: bloquée, car le blocage de $w1(y)$ a bloqué toute la transaction $T1$
- $w2(y)$: acceptée, sous-verrou d'écriture sur y donné à $T2$
- $c2$: acceptée, fin de $T2$ et libération de tous les verrous pris par $T2$

→ déblocage de $w1(y)$ et $c1$ qui sont acceptées

- **Résultat:** $r1(x)$ $r2(y)$ $w2(y)$ $c2$ $w1(y)$ $c1$

– Conflits: $r2[y] - w1[y]$, $w2[y] - w1[y]$ pas de cycle → **sérialisable**

Contrôle de la concurrence

Les verrous : Interblocage

- **Exemple d'ordonnancement** : $r1(x) \ w2(y) \ w2(x) \ w1(y) \ c1 \ c2$

Temps	T1	T2
t1	Lock_S(x)	
t2	Read(x)	
t3		Lock_X(y)
t4		Write(y)
t5		Lock_X(x)
t6		Write(x)
t7	Lock_X(y)	
t8	Write(y)	
t9	commit	
t10		commit

T2 bloquée

T1 bloquée

➔ interblocage ("deadlock") : Provoqué par l'attente circulaire de verrous entre transactions

➔ **Verrou mortel**

Contrôle de la concurrence

Les verrous : Solutions aux Deadlock

Détection du Deadlock :

1. **Durée limite ("timeout"):** annulation d'une transaction qui dépasse la durée limite
 - **Problème:** comment définir la durée limite d'une transaction ?
2. **Graphe d'attente:** représente l'attente de verrous entre transactions
 - Interblocage = cycle dans le graphe d'attente
 - Annulation d'une transaction qui casse le(s) cycle(s)

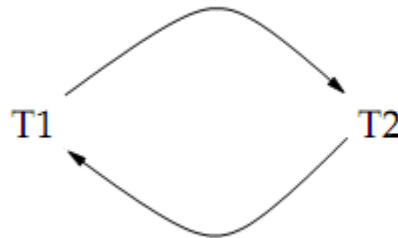
Contrôle de la concurrence

Les verrous : Graphe d'attente

Définition :

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

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



Graphe d'attente de l'exemple 1

Important

- Un verrou indéfini existe ssi le graphe des attentes contient un cycle.

Contrôle de la concurrence

Verrous Mortel : Solutions

Résolution du problème de performances

- Demander la sérialisabilité coûte cher en terme de performances du SGBD
 - beaucoup d'opérations seront mises en attente.
 - Poser des verrous coûte cher
- Si le SGBD attend *sans rien faire la fin d'une transaction* pour poursuivre cela cause un problème de performance!

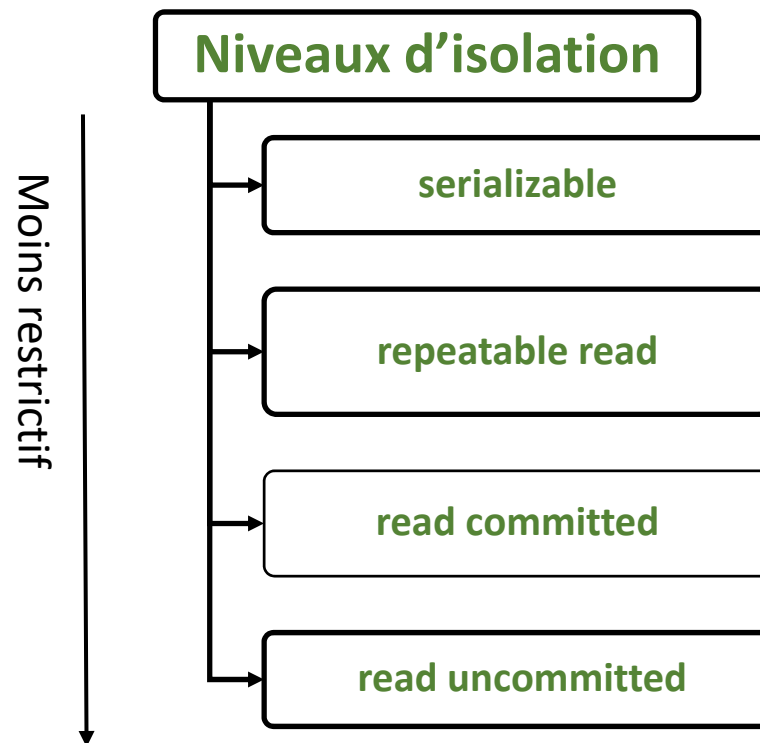
→on pourrait tolérer certaines anomalies :

Degrés ou niveaux d'isolation

Contrôle de la concurrence

Niveaux d'isolation SQL

- Afin de garantir la cohérence, il est évident de toujours choisir un niveau d'isolation maximal, garantissant la sérialisabilité des transactions.
- Le mode **serializable** a l'inconvénient de **ralentir** le débit transactionnel pour des applications qui n'ont peut-être pas besoin de contrôles aussi stricts.
- Ainsi, quatre niveaux d'isolation sont définis :



Contrôle de la concurrence

Niveaux d'isolation SQL

- **serializable** : c'est le mode le plus contraignant, les transactions sont exécutées de façon sérialisable(**séquentiellement**). Il y'a une isolation totale entre les transactions. Une transaction ne voit les modifications d'une autre transaction qu'après la validation des deux transactions.
- **repeatable read** : garantit que le résultat d'une requête est cohérent par rapport à l'état de la base au début de la transaction. La réexécution de la même requête donne toujours le même résultat. La transaction travaille sur un snapshot (ou image) de la BD pris au début de la transaction.
La transaction ne voit que les données validées avant le début de son exécution.
- **read committed** : dans ce cas, une donnée manipulée dans la transaction courante peut être modifiée par une autre transaction, et cette modification peut être vue une fois que cette autre transaction a été validée.
- **read uncommitted** : Ce niveau d'isolation est le plus faible de tous. Il signifie qu'une donnée manipulée dans la transaction courante peut être modifiée par une autre transaction, et que cette modification peut être vue sans que cette autre transaction ait été validée.

Contrôle de la concurrence

Niveaux d'isolation SQL

Problème Toléré Niveau d'isolation	Lecture Impropre	Lecture non reproductible	Tuples fantômes
Read Uncommitted	Possible	Possible	Possible
Read Committed	Impossible	Possible	Possible
Repeatable Read	Impossible	Impossible	Possible
Serializable	Impossible	Impossible	Impossible

Contrôle de la concurrence

Niveaux d'isolation SQL

Choix du niveau d'isolation

- Beaucoup de lectures
- Peu ou pas d'écritures
- Transactions longues
- Peu de transactions



READ COMMITTED
Mode par défaut(Oracle)

- Peu de lectures
- Peu d'écritures
- Transaction courtes
- Beaucoup de transactions



SERIALIZABLE
REPEATABLE READ
Mode par défaut(MySQL)

- Systèmes d'inspection des données (debug)



DIRTY READ