Chapitre 4

Traitements de transactions

4.1 La notion de transaction

Une transaction est une unité atomique de traitement (séquence d'opérations qui doit être exécutée dans son intégralité) qui est : soit complètement exécutée soit complètement abandonnée. Une transaction fait passer la base de données d'un état cohérent à un autre état cohérent. Si une transaction ne va pas à son terme pour une raison ou pour une autre (erreur de syntaxe, violation de contrainte, bug SGBD, arrêt machine, etc), la base est restaurée dans l'état où elle se trouvait avant que la transaction ne démarre.

4.1.1 État cohérent

Définition : Une base de données est cohérente (ou dans un état cohérent) lorsque toutes les données qu'elle contient sont en accord avec les contraintes d'intégrité du schéma conceptuel.

4.1.2 État correct

Définition : Une base de données est dans un état correct lorsqu'elle est dans un état cohérent et que les valeurs des données reflètent exactement le résultat attendu des modifications effectuées.

Exemple :
CREATE TABLE Compte (numero INTEGER PRIMARY KEY, valeur INTEGER) ;

Requêtes d'insertion :
INSERT INTO Compte VALUES (1 , 500) ; INSERT INTO Compte VALUES (2 , 100) ;

Requêtes : Transférer du compte 1 vers le compte 2 la somme 300
R1 : UPDATE Compte SET valeur = valeur - 300 WHERE numero = 1 ;
Q1 : Est-ce que la base de données est dans un état cohérent ?
Q2 : Est-ce que la base de données est dans un état correct ?
R1 : UPDATE Compte SET valeur = valeur + 300 WHERE numero = 2 ;
Entre les deux mises à jour, la base de données est cohérente mais non correcte.

4.2 Les propriétés des transactions

Un système de gestion de transactions doit garantir les propriétés suivantes (ACID) :

a) Atomicité : Une transaction doit effectuer toutes ses mises à jour avec succès ; sinon ne rien faire du tout.

b) Cohérence : une transaction doit laisser la base dans un état cohérent.

c) Isolation : : les modifications effectuées par une transaction ne doivent être visibles aux autres transactions qu'une fois la transaction validée.

d) Durabilité : Dès qu'une transaction valide ses modifications, le système doit garantir que ces modifications seront conservées, même en cas de panne.

4.3 Les ordres des transactions

a) BEGIN TRAN : début de la transaction, en l'absence de cette commande, toute instruction SQL est une transaction implicite.

b) SAVE TRAN : Cette instruction permet de définir des points d'arrêt, et donc donne la possibilité d'annuler une partie de la transaction en cours.

c) COMMIT TRAN : Cette instruction permet de mettre fin avec succès à une transaction.

d) ROLLBACK TRAN : permet d'annuler une partie ou la totalité de la transaction (fin avec échec).

Exemple :
BEGIN TRAN Tr1
- - Ajouter un client
INSERT INTO clients(nom, prenom, telephone) VALUES ('Ali', 'Bal', '021364897') ;
- - Poser un point d'arrêt
SAVE TRAN P1 ;
- - Compter les clients
SELECT COUNT(*) AS NbreClients FROM clients ; NbreClients = 201
DELETE FROM clients ;
SELECT COUNT(*) AS NbreClients FROM clients ; NbreClients = 0
- - Annuler la suppression
ROLLBACK TRAN P1 ;
SELECT COUNT(*) AS NbreClients FROM clients ; NbreClients = 201
- - Valider le reste de la transaction
COMMIT TRAN Tr1 ;
- - Compter les clients
SELECT COUNT(*) AS NbreClients FROM clients ; NbreClients = 201

4.4 Les problèmes de concurrence d'accès

Plusieurs utilisateurs peuvent lancer des transactions en même temps ⇒ Concurrence d'accès. Des transactions exécutées concurremment peuvent interférer et mettre la base de données dans un état incohérent.

Exemple - Problème des accès concurrents

- Considérons deux transactions T1 et T2 qui s'intéressêt à un même objet A
- Les deux seules opérations possibles sur A, sont : lire et écrire

Quatre possibilités :

1. Lecture Lecture
2. Ecriture - Ecriture

3. Ecriture Lecture
4. Lecture Ecriture

1. Lecture-Lecture (Partage)
   — Aucun conflit
   — un même objet peut toujours être partagé en lecture
2. Ecriture Ecriture (Perte de données)
   — T2 peut écraser la valeur de A, par une autre écriture, celle effectuée par T1 (perte de données).

temps	Transaction T1	Transaction T2	Etat de la base
t1	Lire A		A = 20
t2		Lire A	
t3	A := A +50		
t4		A := A +10	
t5	Ecrire A		A = 70
t6		Ecrire A	A = 30

   Résultat : A = 30, Résultat correct : A = 80
3. Ecriture Lecture (Lecture impropre)
   — Une transaction lit des données écrites par une transaction concurrente non validée.
   — T2 lit une valeur modifiée par T1 et ensuite T1 est annulée.

temps	Transaction T1	Transaction T2	Etat de la base
t1	Lire A		A = 20
t2	A := A +50		
t3	Ecrire A		A = 70
t4		Lire A	
t5	Annulation		

   Résultat : A = 70, Résultat correct : A = 20
4. Lecture Ecriture (Lecture non reproductible)
   — T1 modifie la valeur de A entre deux lectures de T2.

temps	Transaction T1	Transaction T2	Etat de la base
t1	Lire A		A = 20
t2		Lire A	
t3	A := A +50		
t4	Ecrire A		A = 70
t5		Lire A	

4.4.1 Sérialisation des transactions

L'état final d'une BD après exécutions en parallèle de transactions doit être identique à une exécution en série des transactions.

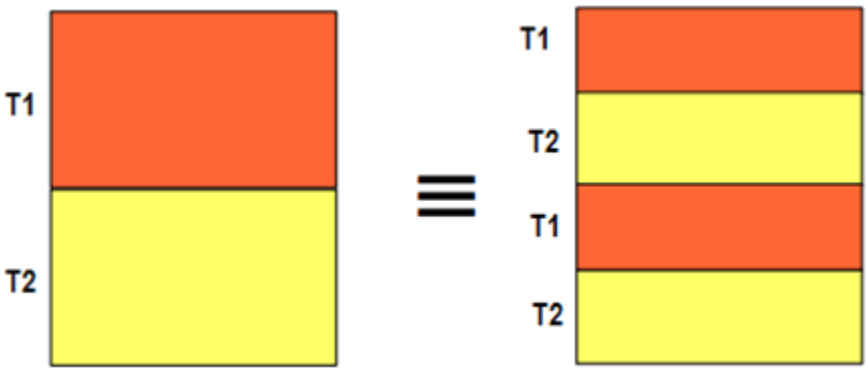

FIGURE 4.1 – Une exécution en parallèle et une exécution en série des transactions

Exemple : Succession : T1,T2

T1	(1) : A := A - 5
	(2) : A := A + 2
T2	(3) : B := B + 3
	(4) : B := B - 6

(2) et (3) sont permutables car elles n'agissent pas sur le même granule (un tuple ou bien une table). On peut transformer en :

(1) : A := A - 5
(2) : B := B + 3
(3) : A := A + 2
(4) : B := B - 6

Cette exécution est sérialisable.

Le problème du contrôle de concurrence consiste pour le système de ne générer que des exécutions sérialisables.

4.4.2 Graphe de précédence

— La notion de précédence de transactions peut être représentée par un graphe.
— Un graphe dont les nuds représentent les transactions et dans lequel il existe un arc Ti vers Tj si Ti précède Tj dans l'exécution analysée.

Exemple :

T1 Lire A ; T2 Ecriture A ; T2 Lire B ; T3 Lire A ; T1 Ecriture B
Pour chaque granule :
A : {LT1, ET2, LT3}
B : {LT2, ET1} (Pas darc entre L et L car on a pas de conflit)
Condition suffisante de sérialisabilité : le graphe de précédence est sans circuit.

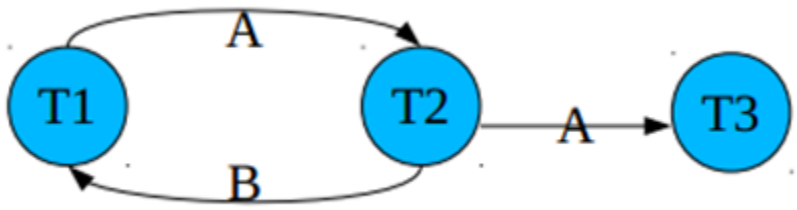

FIGURE 4.2 – Graphe de précédence

— De nombreuses solutions ont été proposées pour traiter le problème des accès concurrents. Deux principales techniques pour garantir la sérialisabilité des transactions :
  — Le verrouillage
  — L'estampillage

4.4.3 Verrouillage

La technique de contrôle des accès concurrents consistent à verrouiller les objets au fur et à mesure des accès par une transaction et à relâcher les verrous seulement après obtention de tous les verrous.

Il repose sur les deux actions :

— Acquérir un verrou sur l'objet A : *verrouiller (A)*
— Relâcher le verrou sur l'objet A : *libérer (A)*

Pour garantir l'isolation des mises à jour, les verrous sont généralement relâchés en fin de transaction. Si une transaction demande un verrouillage d'un objet déjà verrouillé, cette transaction est mise en attente jusqu'à relâchement de l'objet.
un objet A est un granule (un tuple ou bien une table) de la BD.

Deux types de verrous :

— Verrous exclusifs *(X locks)* ou verrous d'écriture
— Verrous partagés *(S locks)*ou verrous de lecture

Verrou demandé	Verrou déjà accordé pour une autre transaction	
	S-lock	X-lock
S-lock	Accordé	Attente de libération
X-lock	Attente de libération	Attente de libération

Protocole d'accès aux données

1. Aucune transaction ne peut effectuer une lecture ou une mise à jour d'un objet si elle n'a pas acquis au préalable un verrou S ou X sur cet objet
2. Si une transaction a un verrou S sur un granule, elle peut demander le verrou X (upgrade).
3. Si une transaction ne peut obtenir un verrou déjà détenu par une autre transaction T2, alors elle doit attendre jusqu'à ce que le verrou soit libéré par T2.
4. Les verrous X sont conservés jusqu'à la fin de la transaction (COMMIT ou ROLLBACK)
5. En général les verrous S sont également conservés jusqu'à cette date.

Phénomènes indésirables

1. **La privation :** .
   — Une transaction risque d'attendre un objet indéfiniment si à chaque fois que cet objet est libéré, il est pris par une autre transaction.
   — Pour traiter ce problème, on peut organiser sur chaque verrou une file d'attente avec une politique "première arrivée", "première servie".

2. **L'inter-blocage (ou verrou mortel) :** Ti attend Tj , Tj attend Ti : il y a inter-blocage.

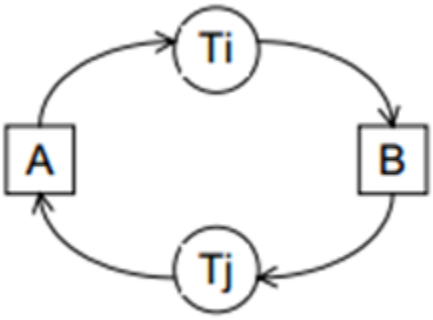

FIGURE 4.3 – L'inter-blocage

Traiter le problème d'inter-blocage

1. **Prévention des inter-blocages :** — Lorsqu'une demande d'acquisition de verrou ne peut être satisfaite on fait passer un test aux deux transactions impliquées, à savoir celle qui demande le verrou, Ti, et celle qui le possède déjà, Tj.
   — Si Ti et Tj passent le test alors Ti est autorisée à attendre Tj, sinon l'une des deux transactions est annulée pour être relancée par la suite.

2. **Détection des inter-blocages :** — Les inter-blocages sont détectés en construisant le graphe "qui attend quoi" et en y recherchant les cycles.
   — Lorsqu'un cycle est découvert l'une des transactions est choisie comme victime, elle est annulée de manière à faire disparaître le cycle.

Syntaxe (MYSQL) : verrous de table

LOCK TABLES nom_table *[READ | WRITE]*;

— En utilisant READ, un verrou de lecture sera posé ; c'est-à-dire que les autres sessions pourront toujours lire les données des tables verrouillées, mais ne pourront plus les modifier.
— En utilisant WRITE, un verrou d'écriture sera posé. Les autres sessions ne pourront plus ni lire ni modifier les données des tables verrouillées.

UNLOCK TABLES ; - - On relâche les verrous

Syntaxe (MYSQL) : verrous de ligne

a- Verrou partagé : Pour poser un verrou partagé, on utilise LOCK IN SHARE MODE à la fin de la requête SELECT.
Exemple de Code SQL
SELECT * FROM WHERE idProduit = 10 *LOCK IN SHARE MODE*;

b- Verrou exclusif : Pour poser un verrou exclusif, on utilise FOR UPDATE à la fin de la requête SELECT.

Exemple de Code SQL

SELECT * FROM WHERE idProduit = 10 *FOR UPDATE*;

Un verrou de ligne est donc lié à la transaction dans laquelle il est posé. Dès que l'on fait un COMMIT ou un ROLLBACK de la transaction, le verrou est levé.

Exemple

On considère l'ordonnancement de T1, T2, T3 suivant :

T1	T2	T3
Lire(A)		
		Ecrire(A)
	Lire(A)	
Ecrire(B)		
	Lire(B)	
		Ecrire(B)

Q1 : Est-ce cet ordonnancement est sérialisable ?

Q2 : Décrivez comment le mécanisme d'accès par verrouillage a deux phases sérialise cet ordonnancement ?

R1 : Graphe de précédence :

T1L(A), T3E(A), T2L(A), T1E(B), T2L(B), T3E(B)

A :{LT1, ET3, LT2}

B :{ET1, LT2, ET3}

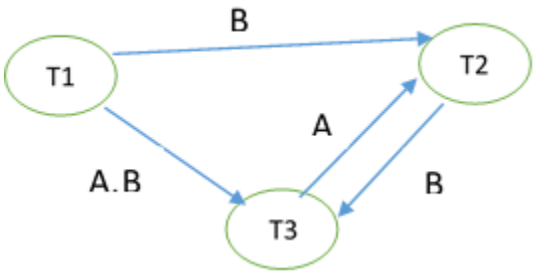

FIGURE 4.4 – Le graphe de précédence de l'exemple

Le graphe avec circuit donc cette exécution peut être sérialisable ou non

R2 : le mécanisme d'accès par verrouillage a deux phases

	T1	T2	T3
t1	S(A)		
t2			Attente X(A)
t3		S(A)	
t4	S(A) X(B)		
t5	Libérer S(A) Libérer X(B)		
t6		S(A) S(B)	
t7		Libérer S(A) Libérer S(B)	
t8			X(A) X(B)
t9			Libérer X(A) Libérer X(B)

Le nouvel ordre dexécution : LT1(A), LT2(A), ET1(B), LT2(B), ET3(A), ET3(B).

A :{LT1, LT2, ET3}

B :{ET1, LT2, ET3}

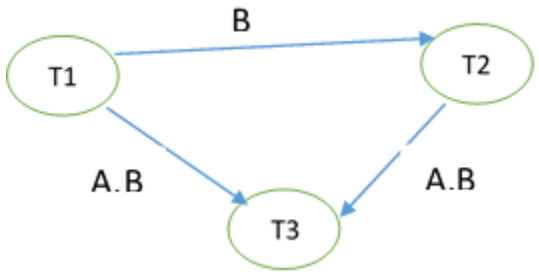

FIGURE 4.5 – Le nouveau graphe de précédence de l'exemple