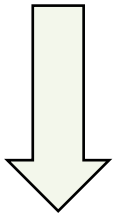
Chapitre 4 : Transactions et gestion de la concurrence d'accès

Introduction aux transactions

- En BD 2 problèmes :
  - De multiples utilisateurs doivent pouvoir accéder à la base de donnée en même temps → problème d'accès concurrents
  - De nombreuses et diverses pannes peuvent apparaître. Il ne faut pourtant pas perdre les données…,



La Gestion de transactions répond à ces problèmes

Notion de transaction

- Une **transaction** est une suite d'opérations interrogeant et/ou modiant la BD, pour laquelle l'ensemble des opérations doit être soit validé, soit annulé.

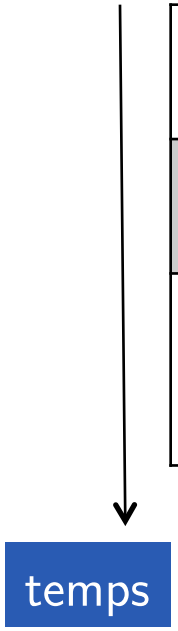
- Toute la transaction est réalisée ou rien ne l'est
  - Validation : toute la transaction est prise en compte
  - Avortement ou annulation : la transaction n'a aucun effet

- Sous oracle
  - Début d'une transaction : ordre SQL ou fin de la précédente
  - Fin d'une transaction : validation(Commit) ou annulation (Rollback)

Notion de transaction

- L'exécution d'une transaction provoque le passage d'un état cohérent de la BD à un nouvel état cohérent

- Une transaction est constituée de trois primitives



Ouverture d'une transaction	**Begin transaction**
Travail sur les données	Travail sur les données
clôture avec confirmation → clôture avec annulation →	COMMIT ROLLBACK

Propriété d'une transaction

- **Atomicité**
  - Soit toutes les modifications effectuées par une transaction sont enregistrées dans la BD, soit aucune ne l'est.
- **Cohérence**
  - Une transaction fait passer une BD d'un état cohérent à un autre état cohérent. Un état cohérent est un état dans lequel les contraintes d'intégrité sont vérifiées.
- **Isolation**
  - Une transaction se déroule sans être perturbée par les transactions concurrentes : tout se passe comme si elle se déroulait seule.
- **Durabilité**
  - Une fois qu'une transaction a été confirmée le SGBD garantit qu'aucune modification qu'elle a effectuée ne sera perdue même en cas : interruption, pannes du système d'exploitation, « crash » de disque, etc.

Vies possibles d'une transcation

- Vie sans histoire
  - La transaction arrive sur la fin
  - Point de confirmation : commit
- Un assassinat
  - Arrêt par un événement extérieur
  - Arrêt par le sgbd lui même (dead lock)
  - Le système fait marche arrière (rollback) et annule les actions déjà effectuées
- Un suicide
  - Arrêt et annulation par la transaction elle même (rollback)

Utilisation des transactions →Accès concurrents

- Accès concurrent
  - Il y a un accès concurrent lorsque plusieurs utilisateurs (transactions) accèdent en même temps à la même donnée dans une base de données.

- Gestion des accès concurrents (contrôle de concurrence)
  - S'assurer que l'exécution simultanée des transactions produit le même résultat que leur exécution séquentielle (l'une puis l'autre)

Accès concurrents

- Problèmes posés par les accès concurrents
  - Perte de mise à jour
  - Lecture impropre
  - Lecture non reproductible
  - Objets fantômes

Problèmes posés par les accès concurrents : Perte de mise à jour

T1 et T2 modifient simultanèment A

T_1	T_2	BD
		$A = 10$
`read A`		
	`read A`	
`A = A + 10`		
`write A`		$A = 20$
	`A = A + 50`	
	`write A`	$A = 60$

Les modifications effectuées par T1 sont perdues

Problèmes posés par les accès concurrents : Lecture impropre

T_1	T_2	BD
		$A + B = 200$
		$A = 120$ $B = 80$
`read A`		
`A = A - 50`		
`write A`		$A = 70$
	`read A`	
	`read B`	
	`display A + B` (150 est affiché)	
`read B`		
`B = B + 50`		
`write B`		$B = 130$

T2 lit une valeur de A non validée, affiche une valeur incohérence

Problèmes posés par les accès concurrents : Lecture impropre (donnée non confirmée)

T_1	T_2	BD
		$A = 50$
	`A = 70`	
	`write A`	$A = 70$
`read A` (70 est lu)		
	`rollback` (La valeur initiale de A est restaurée)	$A = 50$

T1 lit une valeur de A non confirmée

Problèmes posés par les accès concurrents : Lecture non reproductible

T_1	T_2	BD
		$A = 10$
	`read A` (10 est lu)	
`A = 20`		
`write A`		$A = 20$
	`read A` (20 est lu)	

T2 lit deux valeurs de A différentes

Problèmes posés par les accès concurrents : Objet fantôme

T_1	T_2	BD
		$E = \{1, 2, 3\}$
`display card(E)` 3 est affiché		
	`insert 4 into E`	$E = \{1, 2, 3, 4\}$
`display card(E)` 4 est affiché		

Accès concurrents : contrôle des accès

- Verrouillage :
  - Le verrouillage est la technique la plus classique pour résoudre les problèmes dus à la concurrence :
    - Avant de lire ou écrire une donnée une transaction peut demander un verrou sur cette donnée pour interdire aux autres transactions d'y accéder.
    - Si ce verrou ne peut être obtenu, parce qu'une autre transaction en possède un, la transaction demandeuse est mise en attente.
  - Afin de limiter les temps d'attente, on peut jouer sur :
    - la granularité du verrouillage : pour restreindre la taille de la donnée verrouillée (n-uplet, une table)
    - le mode de verrouillage: pour restreindre les opérations interdites sur la donnée verrouillée

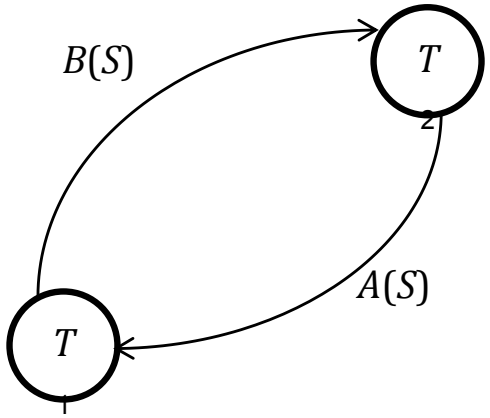
Verrouillage : Exemple

T1	T2	Résultat A=10
Read A avec verrou		
A=A+10	Read A avec verrou	
	attente	A=20
Write A Commit;		A=20
	Read A avec verrou	20
	A=A+50	50
	Write A commit	A=70

Problème de verrouillage : interblocage

L'impasse générée par deux transactions (ou plus) qui attendent, l'une, que des verrous se libèrent, alors qu'ils sont détenus par l'autre

T_1	T_2
`lock X A`	
	`lock X B`
`lock S B`	
attente	`lock S A`
attente	*attente*

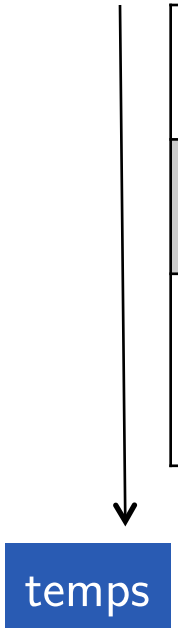


graphe d'attente

Résolution de l'interblocage

- Deux approches
  - Prévention
    - Toutes les ressources nécessaires à la transaction sont verrouillées au départ
    - Problème : cas des transactions qui ne démarrent jamais !
    - Méthode peu utilisée aujourd'hui
  - Détection :
    - On inspecte à intervalles réguliers le graphe d'attente pour détecter si un interblocage s'est produit. Dans ce cas on défait l'une des transactions bloquées et on la relance un peu plus tard.
    - On annule une transaction dont le temps d'attente dépasse un certain seuil, et on la relance un peu plus tard.

Transaction : mise en oeuvre sous oracle



Ouverture d'une transaction	**Connexion ou commit ou rollkack**
Travail sur les données	Travail sur les données
clôture avec confirmation → clôture avec annulation →	COMMIT ROLLBACK

Mise en oeuvre sous oracle

- Début de transaction
  - à la connexion, ou à la fin de la transaction précédente.

- Fin de transaction
  - Commit ou Rollback.
  - Instructions du DDL (CREATE, ALTER, DROP, ..) sont suivies d'un commit implicite, on ne peut donc pas faire d'annulation d'une telle instruction. Si la transaction courante contient des instructions DML, il y a d'abord un commit de ces instructions avant d'effectuer l'instruction DDL.
  - Déconnexion entraîne aussi un commit de la transaction en cours.

Points de repères

- Possible de subdiviser une transaction en plusieurs étapes
  - en sauvant les infos modifiées a la fin de chaque étape,
  - en gardant la possibilité soit
    - de valider l'ensemble des mises a jour,
    - d'annuler une partie des mises a jour a la n de la transaction.
- → Insérer des points de repère, ou SAVEPOINT.

La création d'un point de repère se fait avec l'instruction :
SAVEPOINT pointRepere;

Pour annuler la partie de la transaction depuis un point de repère :
ROLLBACK TO [SAVEPOINT] pointRepere;

Points repères

- Possibilité de défaire jusqu'au point de confirmation

```
[EXEC SQL] insert into employe values(…) ;
[EXEC SQL] savepoint apres_insert ;
[EXEC SQL] delete from employe where … ;
IF %rowcount>50 THEN
[EXEC SQL] rollback to apres_insert ;
[EXEC SQL] commit ;
ELSE
[EXEC SQL] commit ;
END IF
```

Points de repères implicites

- Oracle pose des points de repères (de confirmations) implicites à chaque action : « statement level rollback »

- L'utilisateur peut donc choisir :
  - Effectuer lui même un rollback
  - Tenter de poursuivre la transaction

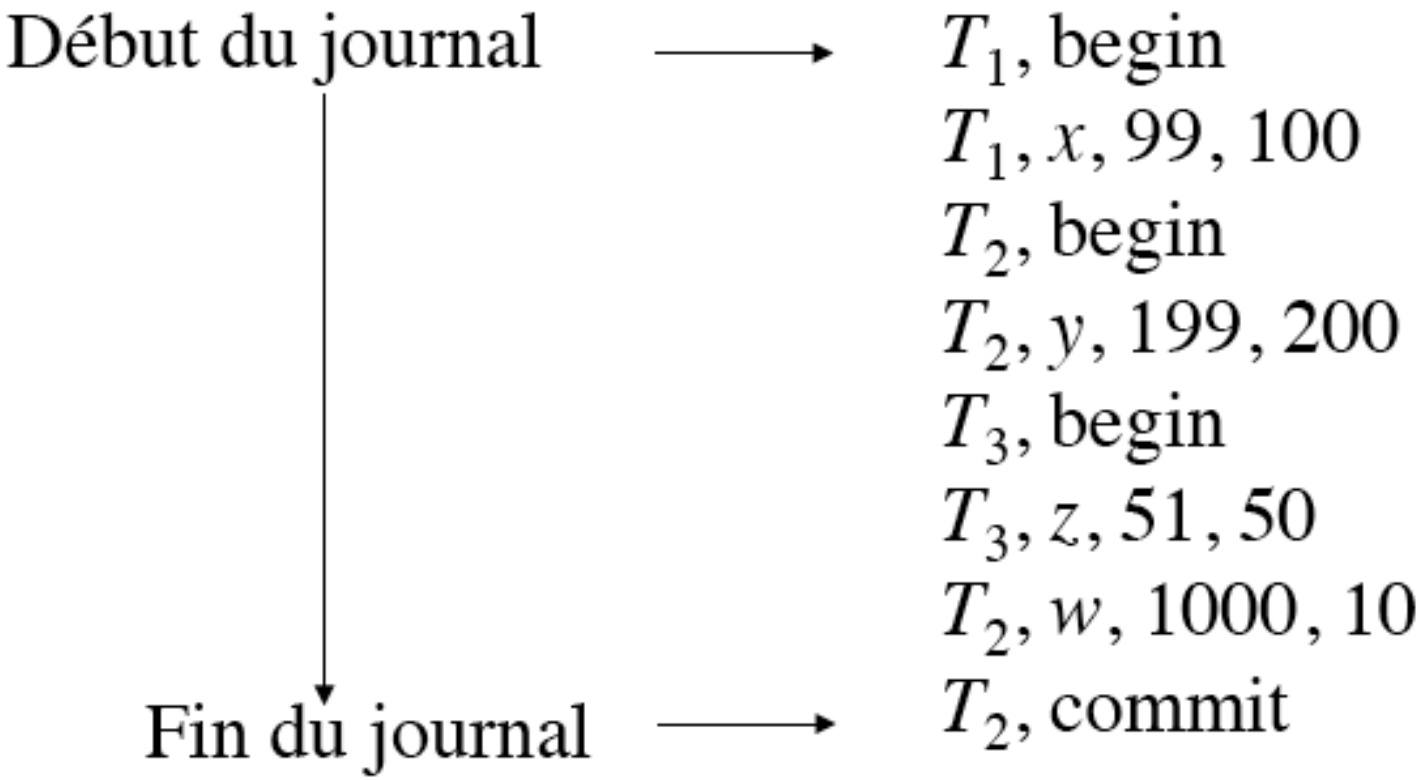
Contrôle de pannes

- Motivations :
  - Environnements non fiables
  - Pannes matérielles
    - Crash Disk,
    - Panne de Courant, ...
  - Pannes logicielles
    - Erreurs dues aux contrôle de concurrence,
    - Violations de contraintes d'intégrité, Crtl C, ...
- Objectifs :
  - Assurer l'atomicité des transactions
  - Garantir la durabilité des effets des transactions commises
- Moyens :
  - Journalisation

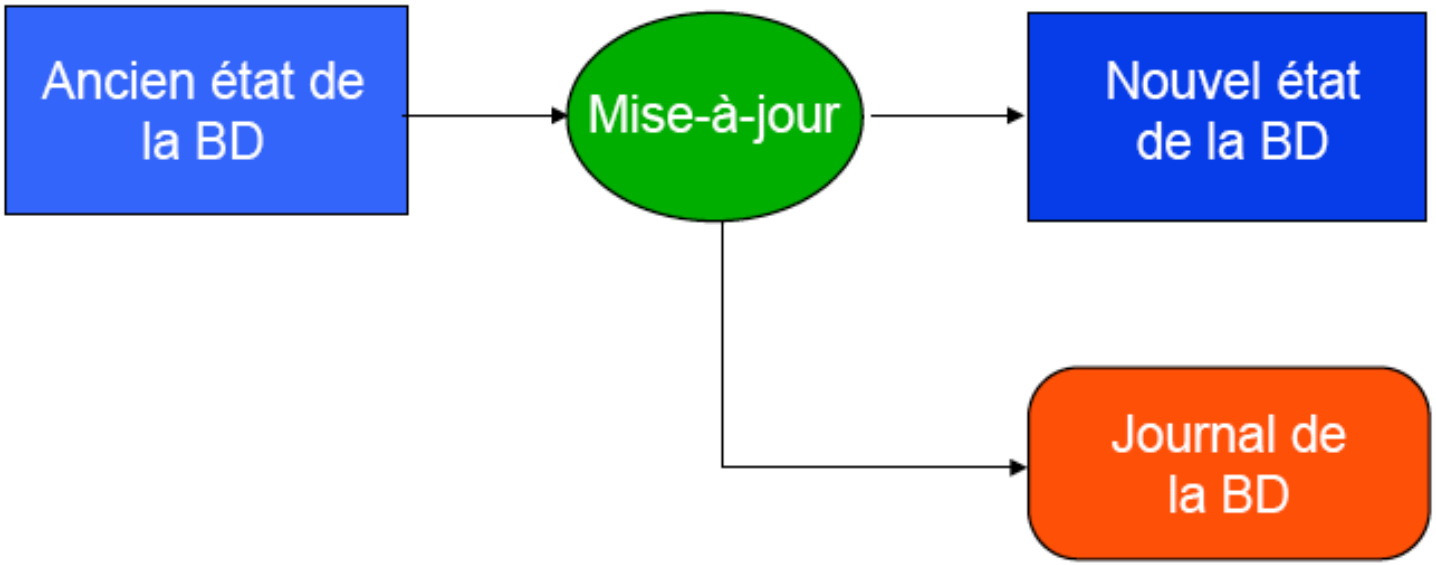
Journalisation des transactions

- Principe général
  - Conservation de la trace des opérations sur la base dans un fichier
  - Plusieurs copies physiquement séparées
- Les informations enregistrées
  - Utilisateur, date , ...
  - Code de l'opération (insert, update, delete)
  - Ancienne valeur – nouvelle valeur
  - commit et rollback
- Les sauvegardes sont notées

Exemple de journal



Journalisation des transactions



Journalisation des transactions

- On peut exprimer l'état de la BD par l'équation :
  - état de la base = journaux de transactions + fichiers de la base

- Règle du point de commit
  - Au moment d'un commit le cache du journal doit être écrit sur le disque. On satisfait donc l'équation : l'état de la base est sur le disque au moment où l'enregistrement commit est écrit dans le fichier journal.
- Règle de recouvrabilité (recovery)
  - Si un bloc du fichier de données, marqué comme modifié mais non validé, est écrit sur le disque, il va écraser l'image avant. Le risque est alors de ne plus respecter l'équation, et il faut donc écrire dans le journal pour être en mesure d'effectuer un rollback éventuel.