12.
Concurrence et transactions

Université de Marne-la-vallée
OlivierCuré *[ocure@univ-mlv.fr]*

Plan

- Notion de transaction
- Théorie de la concurrence
  - Sérialisation
  - Techniques pessimistes et optimistes
- Gestion de transactions
  - Validation
  - Reprise après panne

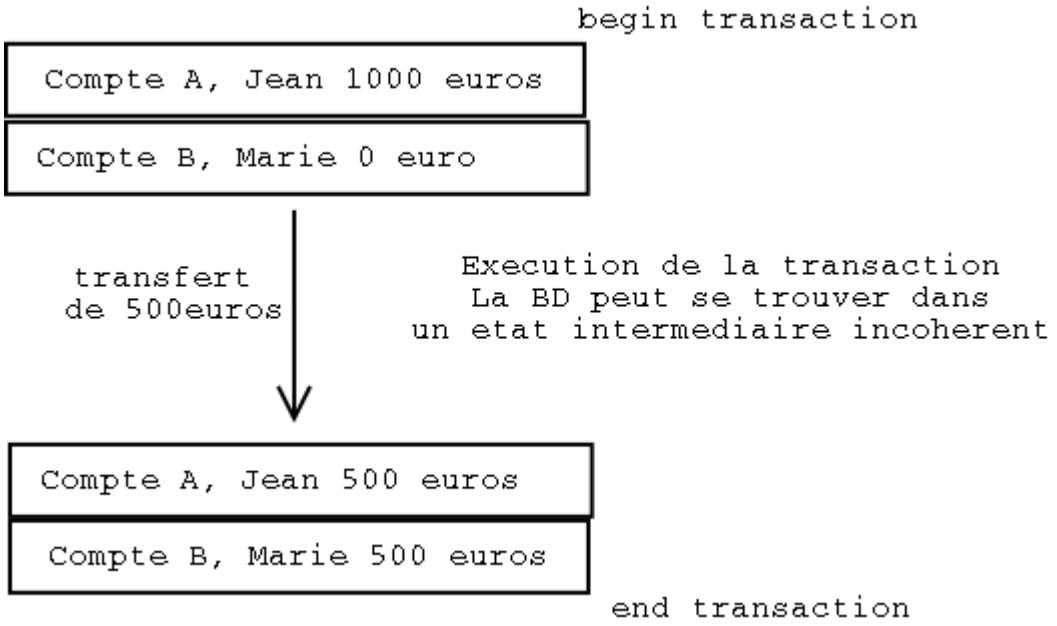
OlivierCuré *[ocure@univ-mlv.fr]*

12.1 Notion de transaction

OlivierCuré *[ocure@univ-mlv.fr]*

Qu'est ce qu'une transaction ?

- C'est une séquence d'opérations (lecture/ écriture) qui doit être exécutée dans son intégralité, amenant la BD d'un état cohérent à un autre état cohérent. Si il n'est pas possible d'accéder à un état cohérent, rien ne doit être fait.



OlivierCuré *[ocure@univ-mlv.fr]*

Propriétés des transactions (ACID)

- **A**tomicité
  - Toutes les mises-à-jour doivent être effectuées ou aucune.
  - En cas d'échec, le système doit annuler toutes les modifications réalisées.
- **C**ohérence
  - La base de données doit passer d'un état cohérent à un autre état cohérent.
  - En cas d'échec, il faut restaurer l'état initial cohérent.

OlivierCuré *[ocure@univ-mlv.fr]*

Propriétés des transactions (ACID)

- **I**solation
  - Les résultats d'une transaction ne sont rendus visibles aux autres transactions qu'une fois celle-ci validée.
  - Les accès concurrents peuvent mettre en question l'isolation.
- **D**urabilité
  - Les modifications d'une transaction validée sont persistentes.
  - Principal problème survient en cas de panne disque.

OlivierCuré *[ocure@univ-mlv.fr]*

12.2 Théorie de la concurrence

Introduction

- La théorie de la concurrence permet de garantir la **cohérence** et l'**isolation** des transactions
- Elle est basée sur la théorié de la **sérialisabilité** des transactions.
- Objectif: rendre invisible aux clients le partage simultané des données.
- Problèmes potentiels: perte de modifications, non-reproductibilité des lectures, modification temporaire.

OlivierCuré *[ocure@univ-mlv.fr]*

Problèmes rencontrés par l'exécution concurrente de transactions

- Problème de **perte des modifications** (*Lost Update*):
  - Scénario : 2 transactions accèdent au même élément d'une BD et l'exécution des opérations est décalée.
  - X=4, Y=8, N=2, M=3

T1: (joe)	T2: (fred)	X	Y
read_item(X);		4	
X:= X - N;		2	
	read_item(X);	4	
	X:= X + M;	7	
write_item(X);		2	
read_item(Y);		8	
	write_item(X);	7	
Y:= Y + N;			10
write_item(Y);			10

Résultat : X=7 et Y=10
Résultat correct :
X= 5 et Y=10

OlivierCuré *[ocure@univ-mlv.fr]*

Problèmes rencontrés par l'exécution concurrente de transactions (2)

- Problème de **non-reproductibilité des lectures** (*Unrepeatable read*):
  - Scénario : 1 transaction réalise une opération d'aggrégation alors qu'une autre transaction est en train de modifier des enregistrements.

T1:	T2:	T1	T2	Sum
	sum:= 0;			0
	read_item(A);		4	
	sum:= sum + A;			4
read_item(X);.	.	4		
X:= X - N;	.	2		
write_item(X);		2		
	read_item(X);		2	
	sum:= sum + X;			6
	read_item(Y);		8	
	sum:= sum + Y;			14
read_item(Y);		8		
Y:= Y + N;		10		
write_item(Y);		10		

Problème : T2 lit X après que N soit soustrait et lit Y avant que N ne soit ajouté.

OlivierCuré *[ocure@univ-mlv.fr]*

Problèmes rencontrés par l'exécution concurrente de transactions (3)

- Problème de **modification temporaire** (*dirty read*):
  - Scénario : 1 transaction va modifier un élément de la BD et la transaction échoue. L'élément mis-à-jour est alors accédé par une autre transaction avant réparation.
  - X= 4 et N=2.

T1: (joe)	T2: (fred)	Database	Log old	Log new
read_item(X);		4		
X:= X - N;		2		
write_item(X);		2	4	2
	read_item(X);	2		
	X:= X- N;	-1		
	write_item(X);	-1	2	-1
failed write (X)		4	rollback T1 log	

OlivierCuré *[ocure@univ-mlv.fr]*

Problèmes rencontrés par l'exécution concurrente de transactions (4)

- Problème de **lecture fantôme** (*phantom read*):
  - Scénario : 1 transaction ajoute un tuple qui satisfait la condition d'une autre transaction.
  - Exemple: une transaction T ajoute un employé travaillant sur le projet 5 alors que la transaction T' demande la liste des employés du projet 5 (pour calculer la somme des heures ou des salaires de ces derniers). Le problème peut être le suivant, T' ne comptabilise pas les données du nouvel employé.

OlivierCuré *[ocure@univ-mlv.fr]*

Granule, Action et Exécution

- Granule de concurrence: unité de données dont les accès sont contrôlés individuellement par le SGBD.
- Dans un SGBD cela peut être un tuple, une page ou bien une table.
- Une action est un accès élémentaire à un granule.
  - Action: unité indivisible par le SGBD sur un granule pour un utilisateur, généralement constituée par une lecture ou une écriture.
- Une suite d'actions est une exécution
  - Exécution: Séquence d'actions obtenues en intercalant les diverses actions des transactions tout en respectant l'ordre interne des actions de chaque transaction.

OlivierCuré *[ocure@univ-mlv.fr]*

Propriétes des opérations sur granule

- Opération: Suite d'actions accomplissant une fonction sur un granule en respectant sa cohérence interne (contraintes d'intégrité).
- Différentes opérations en fonction du granule:
  - Page : lecture et écriture
  - Article (tuple) : lire, écrire, modifier, supprimer et insérer.
- Le résultat d'une opération sur un granule est l'application de l'opération sur le granule
- On distingue les opérations compatibles et permutables

OlivierCuré *[ocure@univ-mlv.fr]*

Opérations compatibles

- Oi et Oj, 2 opérations. Toute exécution simultanée donne le même résultat qu'une exécution séquentielle Eij= Oi suivie de Oj ou de Eji=Oj suivie de Oi. Les résultats de Eij et Eji peuvent être différents.
- O1 et O2 sont compatibles, pas O1 et O3.

```
O1 {
Lire A -> a1
a1+1->a1
Ecrire a1->A
}
```

```
O2 {
Lire B -> b1
b1+1->b1
Ecrire b1->B
}
```

```
O3 {
Lire A -> a2
a2*2->a2
Ecrire a2->A
}
```


OlivierCuré *[ocure@univ-mlv.fr]*

Opérations permutables

- Oi et Oj, 2 opérations. Toute exécution de Oi suivie de Oj donne le même résultat que celle de Oj suivie de Oi.
- O1 et O3 ne sont pas permutables.
- O1 et O4 sont permutables.

```
O1 {
Lire A -> a1
a1+1->a1
Ecrire a1->A
}
```

```
O3 {
Lire A -> a2
a2*2->a2
Ecrire a2->A
}
```

```
O4 {
Lire A -> a1
a1+10->a1
Ecrire a1-> A
}
```


OlivierCuré *[ocure@univ-mlv.fr]*

Opérations compatibles/permutables

- 2 opérations travaillant sur 2 granules différents sont toujours compatibles.
- 2 opérations travaillant sur 2 granules différents sont toujours permutables.
- 2 opérations compatibles sont permutables, mais la réciproque n'est pas vraie.

OlivierCuré *[ocure@univ-mlv.fr]*

Succession et sérialisation

- Une succession de transactions est une exécution E d'un ensemble de transactions (T1,T2,..,Tn) telle qu'il existe une permutation p de (1,2,..n) telle que :
  - E = <Tp(1); Tp(2); ... ;Tp(n)>
- Exécution sérialisable: une exécution E d'un ensemble de transactions (T1,T2,..,Tn) donnant globalement et pour chaque transaction participante le même résultat qu'une succession de (T1,T2,..,Tn).
- Le problème du contrôle de concurrence consiste pour le système de ne générer que des exécutions sérialisables.

OlivierCuré *[ocure@univ-mlv.fr]*

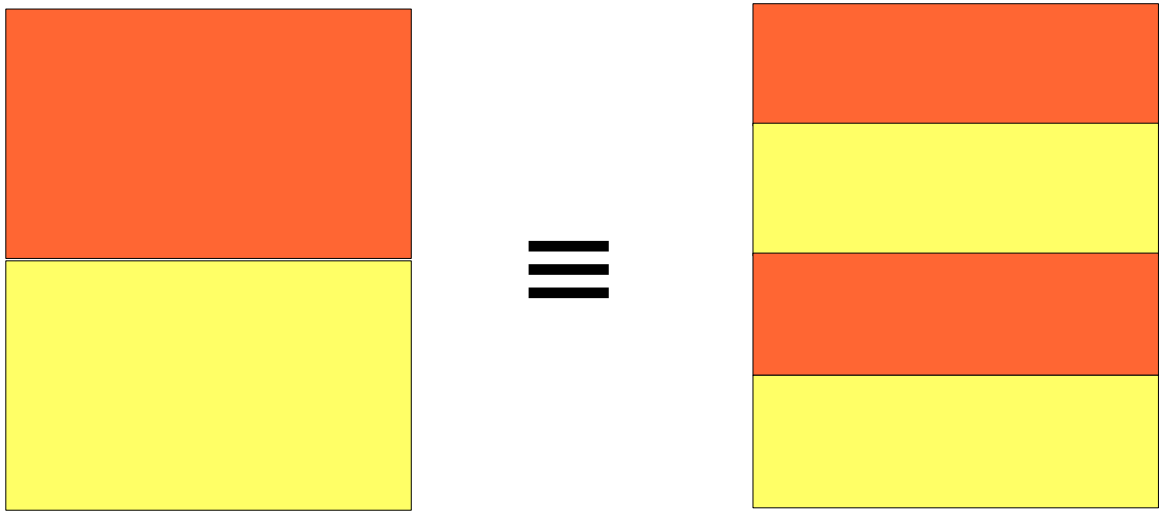
Ordonnancement des transactions

- Exécution en série : une transaction après l'autre.
- Exécutions équivalentes : Quelque soit la BD, l'effet de la première exécution est identique à l'effet de la seconde exécution (moyen de vérification : ordre des lectures et écritures conflictuelles).
- Exécution sérialisable : équivalente à une exécution en série. Si chaque transaction préserve la cohérence, toute exécution en série préserve la cohérence.

OlivierCuré *[ocure@univ-mlv.fr]*

Algorithmes de planification

- 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.



OlivierCuré *[ocure@univ-mlv.fr]*

Sérialisation

- Pour caractériser la sérialisation, 2 transformations sont introduites:
  - Séparation d'opérations compatibles
    - Ex: exécution simultanée de Oi,Oj devient une séquence Oi;Oj ou Oj;Oi.
  - Permutation d'opérations permutables Oi et Oj exécutées par des transactions différentes consiste à changer l'ordre de l'exécution.
- Condition suffisante pour qu'une exécution soit sérialisable: possibilité de transformation par séparation des opérations compatibles et permutations des opérations permutables en une succession de transactions.

Université de Marne-la-vallée
OlivierCuré *[ocure@univ-mlv.fr]*

Sérialisation exemple

(1)T1: A+1-> A

(2)T2: A*2->A (2) et (3) sont permutables

(3)T1: B+1->B car pas sur le même granule

(4)T2: B*2->B

On peut transformer en:

- T1: A+1->A
- T1: B+1->B Cette exécution est sérialisable
- T2: A*2->A
- T2: B*2->B

OlivierCuré *[ocure@univ-mlv.fr]*

Précédence

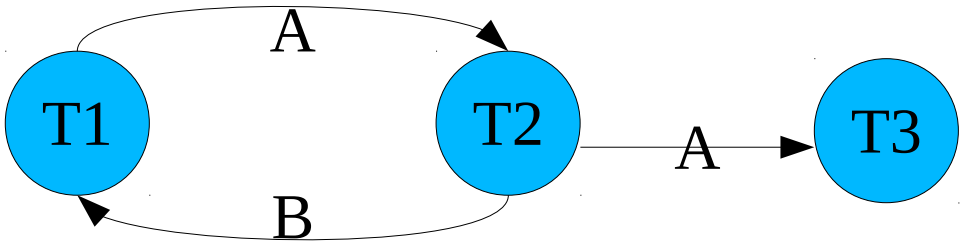
- Une exécution sérialisable est correcte car elle donne le même résultat qu'une exécution l'une après l'autre de transactions.
- Pas toujours possible (ex: non commutativité de l'addition et multiplication).
- Précédence : propriété indiquant qu'une transaction a accompli une opération Oi sur une donnée avant qu'une autre transaction réalise une opération Oj; Oi et Oj n'étant pas commutatives ({Oi;Oj}≠{Oj;Oi}).

OlivierCuré *[ocure@univ-mlv.fr]*

Graphe de précédence

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

{T1:Lire A;
T2: Ecrire A;
T2: Libre B;
T3: Lire A;
T1: Ecrire B}



Condition suffisante de sérialisabilité: le graphe de précédence est sans circuit.

Techniques pour le contrôle de la concurrence

- 2 principales techniques pour garantir la sérialisabilité des transactions:
  - Prévention de conflits, basée sur le verrouillage
  - Détection des conflits, basée sur estampillage

OlivierCuré *[ocure@univ-mlv.fr]*

Contrôle de la concurrence avec technique de blocage

- La technique de **blocage** (**lock**) est la plus populaire dans le SGBD actuels.
- Il s'agit d'une variable associé à un élément de la BD. Généralement, il y a une variable (lock) pour chaque élément de la BD.
- Un lock décrit l'état de l'élément en fonction des opérations possibles (lecture, écriture).
- Cette technique permet de synchroniser les transactions concurrentielles.
- Une transaction bloque un élément avant de l'utiliser
- Lorsqu'une transaction bloque un élément, une autre transaction désirant accéder à cet élément doit attendre son tour.

Université de Marne-la-vallée
OlivierCuré *[ocure@univ-mlv.fr]*

Verrouillage 2 phases

- Technique de contrôle des accès concurrents consistant 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 y a donc 2 phases pour la transaction:
  - Acquisition des verrous
  - Relâchement des verrous.
- Pour garantir l'isolation des mises à jour, les verrous sont généralement relâchés en fin de transaction (verrou long).
- Verrou court: relâchement après exécution de l'opération.

OlivierCuré *[ocure@univ-mlv.fr]*

Verrouillage 2 phases (suite)

- Si une transaction demande un verrouillage d'un objet déjà verrouillé, cette transaction est mise en attente jusqu'à relâchement de l'objet.
- Les circuits sont transformés en **verrous mortels** (*dead lock*).

OlivierCuré *[ocure@univ-mlv.fr]*

Types de blocages

- Blocage binaire avec 2 états distincts :
  - Locked : lock_item(X)
  - Unlocked : unlock_item(X)
- Un mode de blocage multiple permet un accès concurrentiel à un élément par plusieurs transactions. 3 états :
  - Read locked ou shared lock où une autre transaction peut lire l'élément.
  - Write locked ou exclusive lock où une unique transaction bloque l'élément.
  - Unlocked.

OlivierCuré *[ocure@univ-mlv.fr]*

Problèmes liés au blocage

- Deadlock : Chaque transaction attend que l'autre transaction libère un élément.

T1	T2
read_lock(Y)	
read_item(Y)	
	read_lock(X)
	read_item(X)
write_lock(X)	
	write_lock(Y)

OlivierCuré *[ocure@univ-mlv.fr]*

Verrou mortel - Prévention

- Prévention:
  - Priorité sur les transactions
    - Estampille de transaction: numéro unique (timestamp) attribué à une transaction permettant de l'ordonner strictement par rapport aux autres transactions.
    - 2 algos:
      - Wait-Die :si Ti a une priorité supérieure, Ti attend Tj ; autrement Ti abandonne
      - Wound-Die: Si Ti a une priorité supérieure, Tj abandonne ; autrement Ti attend.

OlivierCuré *[ocure@univ-mlv.fr]*

Verrou mortel - Détection

- Détection:
  - Identifier des cycles dans les graphes d'attentes
    - Graphe dont les noeuds correspondent aux transactions et les arcs représentent les attentes entre transactions
    - Si il y a un circuit dans un graphe d'attente, il y a une situation de verrou mortel.

OlivierCuré *[ocure@univ-mlv.fr]*

Gestionnaire de verrouillage

- Les demandes de verrouillage et de déverrouillage sont gérés par le gestionnaire de verrouillage
- La table des verrous pour 1 granule comporte:
  - Nombre de transactions ayant un verrou
  - Type de verrou (shared ou exclusive)
  - Pointeur vers la queue de la file des demandes de verrous
- Verrouillage et déverrouillage sont des opérations atomiques
- Notion d'upgrade d'une verrou: une transaction détenant un verrou partagé peut demander à transformer celui-ci en un verrou exclusif.

Granularité et Verrouillage

- Choix du granule à verrouiller
- Dans un SGBDR, les objets peuvent être : une table, une page ou un tuple.
- Choix d'une unité de verrouillage fine minimise les risque de conflits mais augmente la complexité et le coût du verouillage.

OlivierCuré *[ocure@univ-mlv.fr]*

Degré d'isolation en SQL

- 4 degrés pour l'isolation des transactions, du moins contraignant au plus contraignant:
  - (1)Read uncommitted: verrous courts exclusifs lors de l'écriture.
  - (2)Read commited: verrous longs exclusifs lors de l'écriture.
  - (3)Repeatable read: ajoute la pose de verrous courts partagés en lecture à ceux de degré 2.
  - (4)Serializable: complète le degré 3 avec la pose de verrous longs partagés en lecture.
- Attention aux degrés <4 dans vos applications

OlivierCuré *[ocure@univ-mlv.fr]*

Degré d'isolation en SQL

Degré d'isolation	Dirty Read	Nonrepeatble read	Phantom read
Read uncommitted	possible	possible	possible
Read committed	Pas possible	possible	possible
Repeatable read	Pas possible	Pas possible	possible
Serializable	Pas possible	Pas possible	Pas possible

- Avec PostgreSQL
  - Seulement 2 niveaux avec Postgresql: `serializable` et `read commited`.
  - En sélectionnant `repeatable read`, on obtient en fait `serializable`.
  - Raison: architecture du contrôle de concurrence.

OlivierCuré *[ocure@univ-mlv.fr]*

Contrôle de la concurrence avec détection de conflits

- Le verrouillage est la technique la plus appliquée dans les SGBDR
- D'autres techniques existent comme l'**ordonnancement par estampillage**. Cette technique peut être utilisée pour la résolution de verrous mortels mais aussi pour garantir la sérialisabilité des transactions.
- Principe: conserver pour chaque objet, l'estampille du dernier écrivain W et celle du plus jeune lecteur R.

OlivierCuré *[ocure@univ-mlv.fr]*

Contrôle de la concurrence avec détection de conflits (2)

- Le contrôleur vérifie:
  - que les accès en écriture s'effectuent dans l'ordre croissant des estampilles de transactions par rapport aux opérations créant une précédence, donc l'écrivain W et le lecteur R.
  - Que les accès en lecture s'effectuent dans l'ordre croissant des estampilles de transactions par rapport aux opérations créant une précédence, donc par rapport à l'écrivain W.

OlivierCuré *[ocure@univ-mlv.fr]*

Ordonnancement par estampillage

- Les estampilles W et R associéees à chaque objet remplacent les verrous.
- Problèmes :
  - Reprise de transaction en cas d'accès non sérialisable
  - Risque d'effet domino en cas de reprise de transaction

OlivierCuré *[ocure@univ-mlv.fr]*

Exemple d'une transaction

- `[start_transaction, transaction_id]` : début de l'exécution d'une transaction identifiée par `transaction_id`.
- `[read_item, transaction_id, X]` : la transaction `transaction_id` lit la valeur de l'élément X dans la BD.
- `[write_item, transaction_id, X,old_value, new_value]` : la transaction `transaction_id` change la valeur de l'élément X d'une ancienne à une nouvelle valeur.

OlivierCuré *[ocure@univ-mlv.fr]*

Exemple d'une transaction (2)

- `[commit, transaction_id]` : la transaction `transaction_id` a effectué toutes les opérations avec succès et les effets peuvent être enregistrés dans la BD.
- `[abort, transaction_id]` : la transaction `transaction_id` n'a pas été complétée suite à une problème.

OlivierCuré *[ocure@univ-mlv.fr]*

Etats d'une transaction

- Begin transaction : indique le début d'une transaction.
- End transaction : indique la fin d'une transaction. Les opérations d'écriture et lecture se sont terminées correctement.
- Commit transaction : signale le succès de la transaction. Les modifications sont enregistrées dans la BD et ne peuvent être annulées.
- Rollback : signale l'échec de la transaction. Les modifications sur la BD doivent être annulées.

OlivierCuré *[ocure@univ-mlv.fr]*

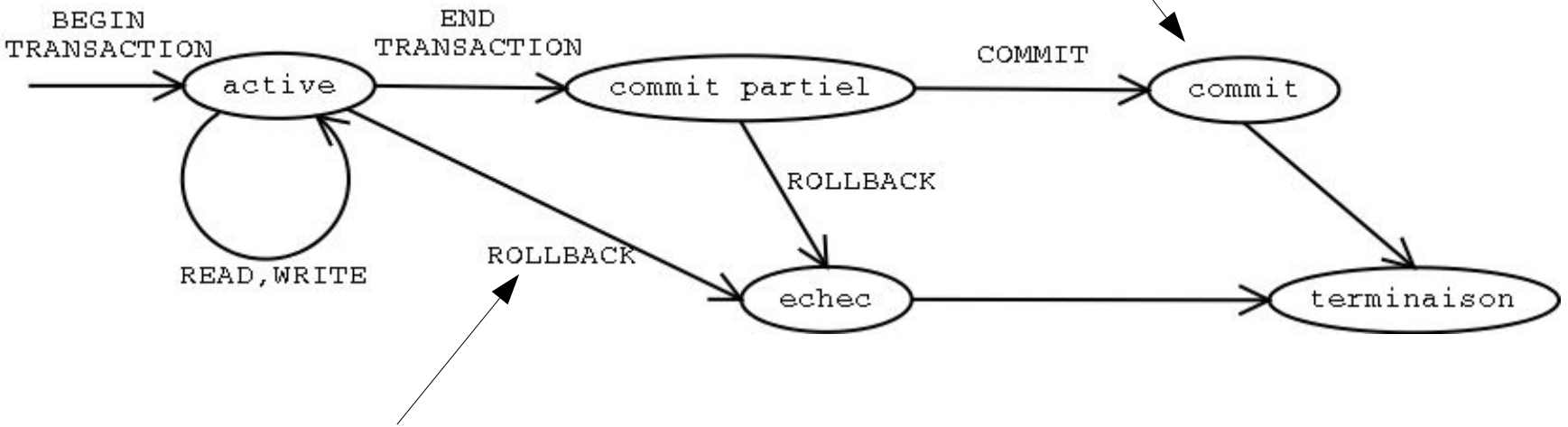
Etats d'une transaction (2)

- Undo : similaire à un ROLLBACK mais s'applique à une unique opération plutôt qu'à une transaction.
- Redo : va spécifier que certaines opérations d'une transaction doivent être rejouées.

OlivierCuré *[ocure@univ-mlv.fr]*

Exécution d'une transaction

Une transaction accède à un état de commit lorsque toutes les opérations sur la BD ont été complétées et que le résultat a été enregistré dans le log.



Lors d'une panne, il faut chercher dans le log et annuler les modifications effectuées, les opérations [write] sans [commit].

OlivierCuré *[ocure@univ-mlv.fr]*

Concurrence et recouvrement

- Contrôle de la concurrence
  - Les SGBD sont multi-utilisateurs.
  - Exécution concurrente de plusieurs transactions soumises par plusieurs utilisateurs :
    - est organisée de telle sorte que chaque transaction n'interfère pas avec les autres transactions en produisant des résultats incorrects.
    - Doit donner l'impression que les transactions s'exécutent de manière isolée.
- Recouvrement
  - Les pannes du système (logiciels ou matériels) ne doivent pas laisser la BD dans un état incohérent.

OlivierCuré *[ocure@univ-mlv.fr]*

Transaction comme unité de recouvrement

- Si une erreur survient (logiciel/matériel) entre le début et la fin d'une transaction, la BD risque de devenir incohérente.
  - Crash du serveur de BD.
  - Erreur de la transaction.
  - Erreur locale ou exception levée par la transaction.
  - Panne disque.
- La BD doit être restaurée dans un état le plus proche possible d'avant panne.

OlivierCuré *[ocure@univ-mlv.fr]*

Transaction comme unité de recouvrement (2)

- Le SGBD doit assurer qu'en cas de panne sur une transaction comportant des opérations de modification, celles-ci soient annulées.
- Les opérations de **COMMIT** et **ROLLBACK** (ou équivalents) supportent l'atomicité de la transaction.

OlivierCuré *[ocure@univ-mlv.fr]*

Solution pour le recouvrement

- Mirroir
  - Garder 2 copies de la BD et les maintenir simultanément.
- Sauvegarde
  - Réaliser périodiquement un "dump" complet de l'état de la BD sur un support tertiaire.
- Système de logging
  - Garder un log de toutes les opérations d'affectations des données des transactions. Le log est sauvegardé sur disque.

OlivierCuré *[ocure@univ-mlv.fr]*

Recouvrement après erreur sur transaction

- Panne catastrophique
  - Implanter une version de la BD depuis une copie de sauvegarde.
  - Exploiter le log des transactions pour reconstruire un état cohérent le plus proche possible d'avant panne.
  - Exploiter le dump plus le log des transactions
- Panne non catastrophique
  - Annuler les opérations qui ont provoquées l'incohérence.
  - Les entrées du log sont consultées lors de la réparation.
  - Pas besoin d'utiliser une copie complète de sauvegarde.

OlivierCuré *[ocure@univ-mlv.fr]*

Conclusion

- Des techniques complexes
- Problème bien maîtrisé dans les SGBDR
- La concurrence complique la gestion des transactions
- Les transactions lognues restent problèmatiques
- Enjeu essentiel pour le commerce électronique
  - Validation fiable
  - Reprise et copies
  - Partage de connections
  - Partage de charge

OlivierCuré *[ocure@univ-mlv.fr]*