Agenda

- Mécanisme de transactions des SGBD
  - Transactions, propriétés ACID
  - Gestion de la concurrence, verrous
  - Reprise après erreur
- Principes des systèmes répartis appliqués aux bases de données
  - Répartition, duplication

Sommaire

- Rappels
- Transactions
- Propriétés ACID
  - “Atomicité”, “Cohérence”, “Isolement”, “Durabilité”
- Concurrence
  - Entrelacement, Sérialisabilité
  - Verrous (Locks)
  - Etreinte fatale (Deadlock)
- “Recovery” (recupération des erreurs)
  - Commit / Abort / Rollback / Checkpoints

Rappel: Table

Table

- **Schéma**: NomTable(Attribut1, Attribut2, ..)
- **Record/Tuple**: une entrée dans la table
- **Element**: une valeur d'une entrée dans la table

Clés: attribut avec valeur unique

- NomTable(Attribut1, Attribut2, ..)

Rappel: Opérations

- Créer / Altérer / Détruire des tables tables et leurs attributs
- Requêtes sur une ou plusieurs tables
- Insérer / Détruire / Modifier des Tuples dans les tables

Rappel: Opération

Créer / Détruire / Modifier schéma des tables

```
CREATE TABLE NomTable
Attribut1 TypeAttribut1
Attribut2 TypeAttribut2
..

DROP NomTable

ALTER TABLE NomTable
   ADD Attribut3 TypeAttribut3
```

Rappel: Opération

Requêtes / Query (sur une ou plusieurs tables existantes):

SELECT attributes

FROM relations (multiple, join)

WHERE conditions (selections)

Modifier une ou plusieurs entrée (tuples) dans une table: Insert, Delete, Update

INSERT INTO R(A1,…., An) VALUES (v1,…., vn)

DELETE FROM NomTable

WHERE conditions

UPDATE NomTable

SET atttribut = valeur

WHERE conditions

Rappel: Règles d'intégrité

- Clés -> unicité des valeurs de la clé
- Attributs -> conditions
- Tuple (record) -> conditions
- Global (assertions) -> à travers plusieurs tables

Contraintes d'intégrité (voir lecture 2)

- CI Structurelles
  - Valeurs par défauts, clés primaires
- CI Référentielles
  - Clés secondaires
- CI de Domaine
  - Contraintes sur les attributs
  - Contraintes sur les tuples
- Triggers (Event-Condition-Action)

Rappel: SGBD

- Query engine
- Query optimisation
- Gestion du stockage
- **Gestion des Transactions (concurrence, récupération après erreur)**

Transaction

Transaction = séquence d'opérations qui

soit réussissent toutes,

soit échouent toutes

Transaction = programme qui transforme une base de données d'un état cohérent à un autre état cohérent

Exemple

1: Begin_Transaction

2: get (K1, K2, CHF) from terminal

3: **Select BALANCE Into S1 From ACCOUNT Where ACCOUNTNR = K1;**

4: S1 := S1 - CHF;

5: **Update ACCOUNT Set BALANCE = S1 Where ACCOUNTNR = K1;**

6: **Select BALANCE Into S2 From ACCOUNT Where ACCOUNTNR = K2;**

7: S2 := S2 + CHF;

8: **Update ACCOUNT Set BALANCE = S2 Where ACCOUNTNR = K2;**

9: **Insert Into BOOKING(ACCOUNTNR,DATE,AMOUNT,TEXT)**

Values (K1, today, -CHF, 'Transfer');

10: **Insert Into BOOKING(ACCOUNTNR,DATE,AMOUNT,TEXT)**

Values (K2, today, CHF, 'Transfer');

12: If S1<0 Then **Abort_Transaction**

11: **End_Transaction**

Problèmes

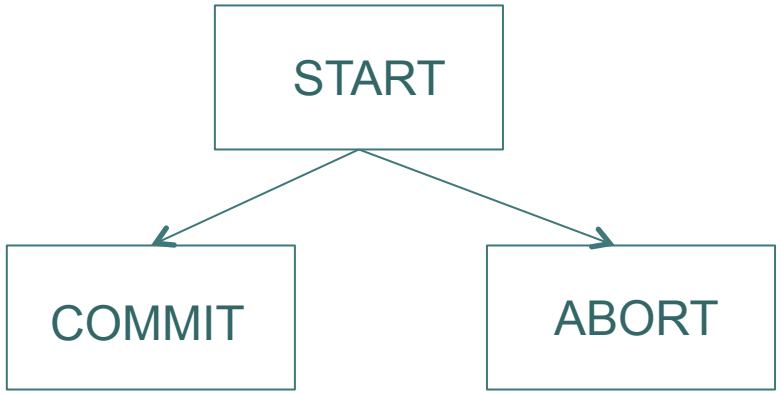
Si plusieurs transactions sont exécutées en même temps

- D'autres applications ont accès à un état intermédiaire inconsistent
- Solution: **contrôle de la concurrence**
- Exemple: 10 clients utilisent le même serveur en parallèle

Système "crashe" (a une panne) pendant une transaction

- Base de données reste dans un état intermédiaire inconsistent
- Solution: **recovery / recupération après erreur**

Transaction



Transaction

Une transaction peut se terminer

- normalement (après un *commit*)

  Elle est alors validée. Toutes les modifications de la base de données sont enregistrées et la base de données est cohérente. Le système ne reviendra pas en arrière.

- de manière avortée (après un *abort*)

  Elle ne peut pas être menée au bout. La base de données est purgée des modifications effectuées par cette transaction et ramenée à l'état dans lequel elle se trouvait avant début de la transaction

Propriétés ACID

Atomicité

Toutes les opérations sont exécutées ou aucune ne l'est (« tout ou rien »)

Cohérence

Une transaction valide les règles d'intégrité de la BD (état consistent avant et après la transaction)

Isolation

Aucune mise-à-jour n'est visible par les autres transactions avant la validation (commit). Indépendence des transactions.

Durabilité

Les changements réalisés par la transaction sont enregistrés de manière permanente après la validation.

Gestion des Transaction

- Le SGBD vérifie les propriétés ACID
- Exemples d'interruption de transaction:
  - violation de règle d'intégrité
  - panne
  - arrêt par intervention d'un responsable
  - étreinte fatale (en cas de transactions concurrente)

Gestion des Transactions

- Isolation + Cohérence => Contrôle de la Concurrence
  - Transactions concurrentes apparaissent comme si elles étaient exécutées séquentiellement

- Atomicité + Durabilité => Recovery

Modèle des Transactions

Chaque transaction lit/écrit un ou plusieurs “éléments” d’une base de donnée

- Un block (schéma)
- Une relation
- Un tuple (entrée)

Contrôle de la concurrence

Concurrence améliore la performance

"Interleaving" des opérations des différentes transactions donne lieu à des anomalies

Problèmes typiques/classiques

- Perte d'une mise à jour (update)
- Lecture "sale" (dirty read)
- Lecture que l'on ne peut pas reproduire

Perte d'une mise à jour

T1, T2 déposent de l'argent sur le compte acc1

Transactions		State
T1:	T2:	acc1
read(acc1)		20
acc1 := acc1 + 10	read(acc1)	20
	acc1 := acc1 + 20	
	write(acc1)	40
	commit	
write(acc1)		30
commit		

Changements dus à T2 sont perdus:

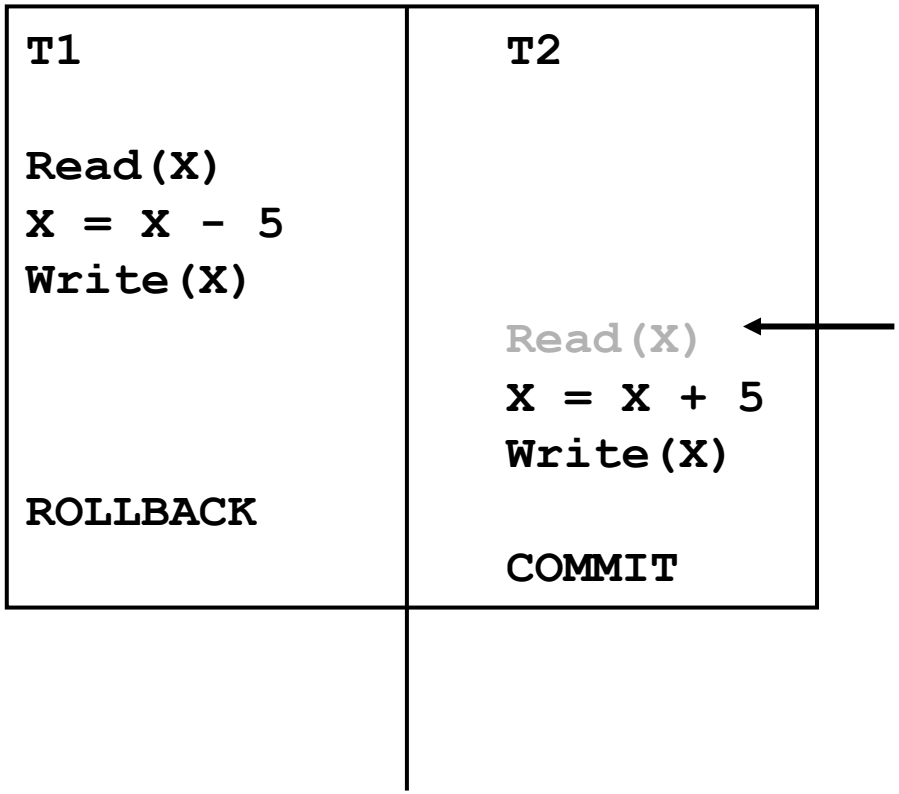
R1(acc1) R2(acc1) W2(acc1) W1(acc1)

Perte d'une mise à jour

Règle : Toute transaction T2 ne doit pas écrire sur un objet dont la valeur a été modifiée par une autre transaction T1 qui n'a pas été validée.

(write / write)

Uncommitted Update (“dirty read”)



Lit la valeur de X alors qu’il n’aurait pas dû la voir

“Dirty” read

T1 dépose deux montants sur le compte acc1

T2 fait la somme de tous les comptes

Transactions		State	
T1:	T2:	acc1	sum
read(acc1)		20	0
acc1 := acc1 + 10			
write(acc1)	...		
	read(acc1)	30	
	sum := sum + acc1		
	write(sum)		30
acc1 := acc1 + 10	commit		
write(acc1)		40	
commit			

T2 voit les valeurs intermédiaires (sales) de T1

R1(acc1) W1(acc1) R2(acc1) W2(sum) W1(acc1)

“Dirty” Read

Règle : Toute transaction T ne doit pas lire des valeurs non confirmées et manipulées par d’autres transactions

(write / read)

Lecture non reproductible

T1 lit plusieurs fois le compte acc1

T2 dépose un montant sur le compte acc1

Transactions		State
T1:	T2:	acc1
read(acc1)		20
	read(acc1)	20
	acc1 := acc1 + 20	
	write(acc1)	40
	commit	
read(acc1)		
sum := sum + acc1		
write(sum)		40
commit		

T1 lit des valeurs différentes de acc1

R1(acc1) R2(acc1) W2(acc1) R1(acc1) W1(sum)

Lecture non reproductible

Règle : Aucune transaction ne peut modifier une valeur lue par une autre transaction avant que cette dernière ne soit validée.

(read / write)

Gestion de la concurrence

- Sérialisabilité
  - Graphe de Sérialisabilité
- Verrouillage (Locks)
  - Protocole de verrouillage à deux phases
  - Etreinte fatale (Deadlock) – Problème des philosophes

Sérialisabilité

Schedule = entrelacement (interleaving) des actions (read/write) d'un ensemble de transactions, où les actions de chaque transactions sont dans l'ordre d'origine

Schedule complet = avec les commit/abort

Sérialisabilité

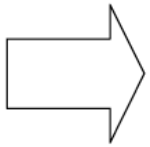
Schedule Complet:

Etat initial de BD + Schedule complet -> état final de la BD

```
Transactions
T1:                 T2:
read(acc1)

                    read(acc1)
                    acc1 := acc1 + 20
                    write(acc1)
                    commit

read(acc1)
sum := sum + acc1
write(sum)
commit
```



```
Schedule

read1(acc1)

read2(acc1)

write2(acc1)
commit2
read1(acc1)

write1(sum)
commit1
```

Schedule Sérialisé

Une transaction à la fois, sans entrelacement

```
T1:                     T2:
                        read(acc1)
                        acc1 := acc1 + 20
                        write(acc1)
                        commit

read(acc1)
read(acc1)
sum := sum + acc1
write(sum)
commit
```

Etat final est consistent

Résultat peut être différent si la séquence des transactions est différente

Schedule Sérialisable

Schedule sérialisable est un schedule avec des transactions entrelacées, s'il existe un schedule sérialisé qui produit le même résultat

Dans un schedule sérialisé, les transactions sont isolées

Sérialisabilité

T1	T2
r1(a)	
	r2(a)
w1(a)	
	r2(a)
r1(b)	
	r2(b)

Peut-on trouver un schedule sérialisé?

Graphe de Précédence

Graphe de de précédence

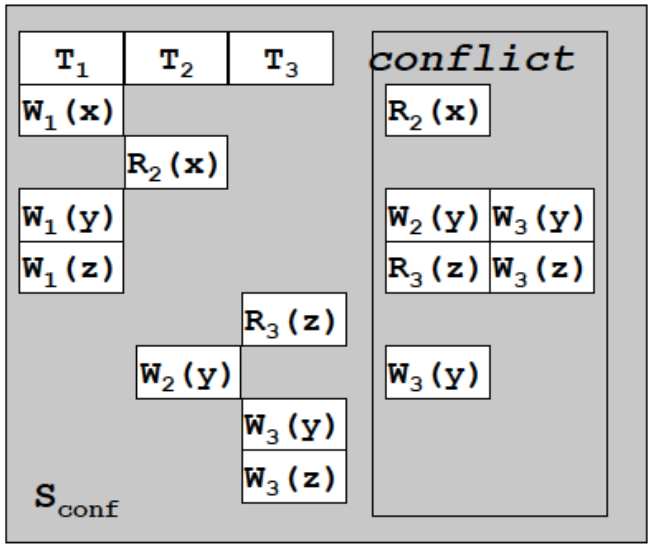
- nœuds = transactions
- arcs = précédences
- la transaction Ti précède Tj (Ti -> Tj) dans le graphe si il y a une action de Ti qui précède et est en conflit avec une action de Tj:
  - Ti et Tj s'exécutent de manière concurrente,
  - Tj modifie/écrit un objet que Ti a déjà lu ou écrit,
  - ou Tj lit un objet que Ti a déjà modifié/écrit
  - (i.e. il y a deux actions sur un même objet, l'une des deux actions au moins est un write)
- Si le graphe a un cycle alors le schedule n'est pas sérialisable
- Si le graphe n'a pas de cycle : n'importe quel ordre topologique des transactions peut être pris comme schedule sérialisé.

Sérialisabilité

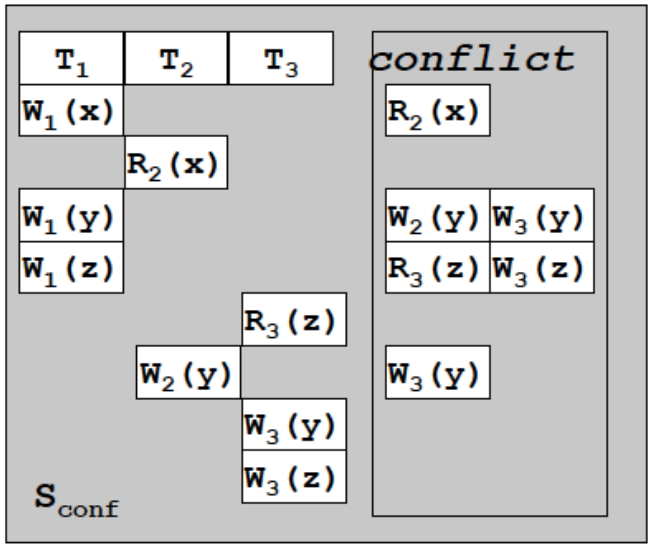
T1	T2
r1(a)	
	r2(a)
w1(a)	
	r2(a)
r1(b)	
	r2(b)

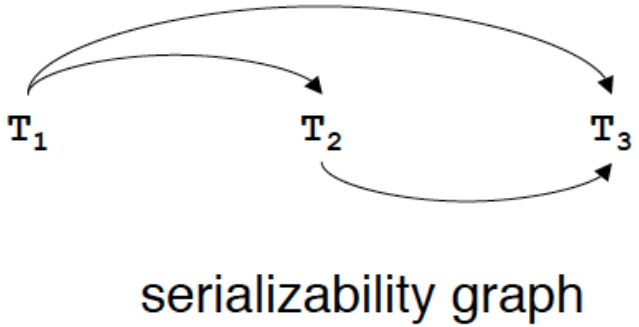
Peut-on trouver un schedule sérialisé?

Graphe de Précédence



Graphe de Précédence





serializability graph

Vérification de la sérialisabilité

Optimiste: sérialisabilité est vérifiée après que les transactions soient effectuées en utilisant le graphe de sérialisabilité, sinon avorter les transactions

Pessimiste: éviter la non-sérialisabilité avant quelle ne se produise – utiliser des verrous

Résumé

- Transactions
- Propriétés ACID
- Transactions concurrentes et problèmes
- Gestion de la concurrence
  - Sérialisabilité
  - Verrous

Lectures Recommandées

[Lewis] Philip M. Lewis, Arthur Bernstein, Michael Kifer: Databases and Transaction Processing - An application-oriented approach, Addison-Wesley.