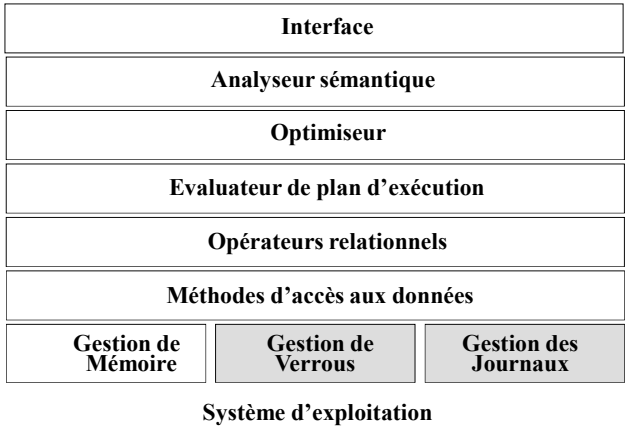
Gestion de Transactions

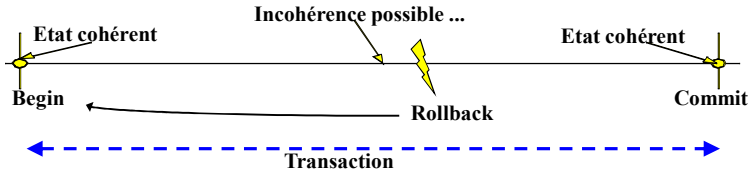
- Propriétés ACID d'une transaction
- Protocoles de contrôle de concurrence
- Protocoles de tolérance aux pannes

Architecture en couche d'un SGBD



Définition d'une transaction

Transaction = séquence d'instructions d'un programme de base de données faisant passer la base d'un état cohérent à un état cohérent.



Begin
CEpargne = CEpargne - 3000
CCourant = CCourant + 3000
Commit

Aucune opération ne peut être exécutée hors transaction (*transactions chaînées*)
Une transaction peut se terminer par une validation (Commit) ou un abandon (Rollback)

Propriétés ACID d'une transaction

Atomicité
Une transaction doit s'exécuter en tout ou rien

Cohérence
Une transaction doit respecter l'ensemble des contraintes d'intégrité

Isolation
Une transaction ne 'voit' pas les effets des transactions concurrentes

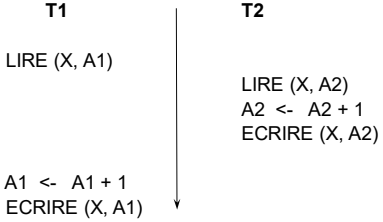
Durabilité
Les effets d'une transaction validée ne sont jamais perdus, quel que soit le type de panne

On parle de contrat transactionnel
entre l'utilisateur/programme et le SGBD

Accès concurrent: les problèmes (1)

Remarque : les transactions concurrentes s'exécutent dans des threads/processus différents

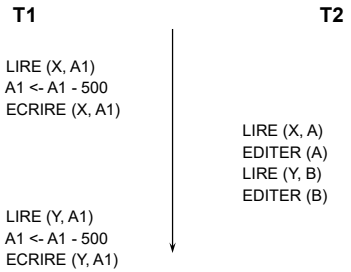
Perte d'opération



Accès concurrent: les problèmes (2)

Observation d'incohérences

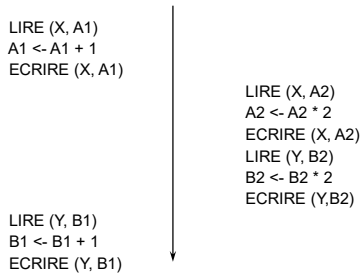
Contrainte d'Intégrité : X = Y



Accès concurrent: les problèmes (3)

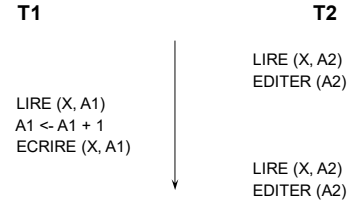
Introduction d'incohérences

Contrainte d'Intégrité : X = Y



Accès concurrent: les problèmes (4)

Lectures non reproductibles



Exécution sérialisable

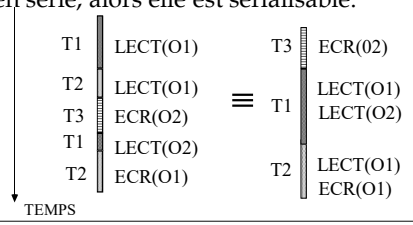
• **Exécution sérialisable** : une exécution en parallèle des transactions T1, ..., Tn est dite sérialisable si son résultat est équivalent à une exécution en série quelconque de ces mêmes transactions.

• Les actions A et B sont **commutables/permutables** si toute exécution de A suivie par B donne le même résultat que l'exécution de B suivie par A.

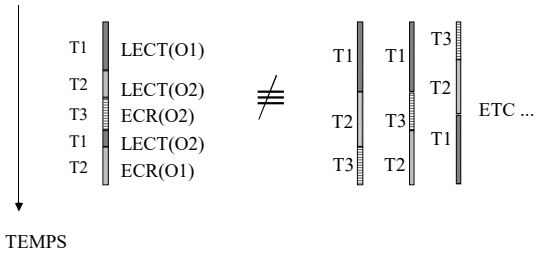
• Si une exécution de transactions est transformable par commutations successives en une exécution en série, alors elle est sérialisable.

Exemple

d'exécution sérialisable



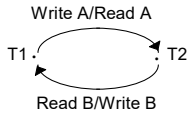
Exemple d'exécution non sérialisable



Relation de précédence

• Ti précède Tj dans une exécution s' il existe deux actions non permutables ai et aj tel que ai soit exécutée par Ti avant que aj soit exécutée par Tj

Exemple :



• Si le graphe de précédence d'une exécution est **sans circuit**, l'exécution est **sérialisable.**

VERROUILLAGE 2 PHASES

OBJECTIF

LAISSER S'EXECUTER SIMULTANEMENT SEULEMENT LES OPERATIONS COMMUTABLES

	Lecture	Ecriture
Lecture	**1**	**0**
Ecriture	**0**	**0**

MOYEN

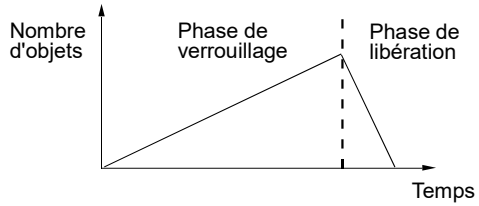
— LES TRANSACTIONS VERROUILLENT LES OBJETS AUXQUELLES ELLES ACCEDENT

— LES TRANSACTIONS CONFLICTUELLES SONT MISES EN ATTENTE (PLUTOT QUE D'ETRE ABANDONNEES)

Condition de Bon Fonctionnement

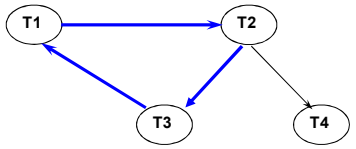
n **TOUTE TRANSACTION DOIT ETRE COMPOSEE DE DEUX PHASES :**

==> 2PL (Two Phase Locking)



 LE DEVERROUILLAGE S'EFFECTUE EN FIN DE TRANSACTION AFIN D'EVITER QU'UNE TRANSACTION PUISSE VOIR DES MISES A JOUR NON VALIDEES (DIRTY READS)

Problème du verrou mortel



• **Détection**

on construit le graphe des attentes au fur à mesure des conflits. Si un cycle est détecté, on abandonne une des transactions du cycle

Degrés d'isolation SQL

- **Objectif: accroître le parallélisme en autorisant** certaines transactions **à violer la règle d'isolation**
- **Degrés standardisés SQL2 (Set Transaction Isolation Level)**
  - **Degré 0 (Read Uncommitted)**
    - » Lecture de données sales – Interdiction d'écrire.
    - » *Ex. lecture sans verrouillage*
  - **Degré 1 (Read Committed)**
    - » Lecture de données propres mais non reproductible – Ecritures autorisées
    - » *Ex. Verrous court en lecture, long en écriture*
  - **Degré 2 (Repeatable Read)**
    - » Pas de lecture non reproductible
    - » *Ex. Verrous longs en lecture et en écriture*
  - **Degré 3 (Serializable)**
    - » Pas de requête non reproductible (fantôme)

Degrés d'isolation SQL : exemples

T1(°0, Read uncommitted)	**T2(°3,serializable)**	**T1(°1, Read committed)**	**T2(°3,serializable)**
Begin Lit CC (→ 100) Lit CC (→ 110) Lit CC (→ 130)	Begin Lit CC (→ 100) Ecrire CC, CC+10 Ecrire CC, CC+20 Commit	Begin Lit CC (→ 100) Lit CC (→ bloque) (→ 130)	Begin Lit CC (→ 100) Ecrire CC, CC+10 Ecrire CC, CC+20 Commit

T1(°2, Repeatable read)	**T2(°3,serializable)**	**T1(°2, Repeatable read)**	**T2(°3,serializable)**
Begin Lit CC (→ 100) Lit CC (→ 100) Lit CC (→ 100) Commit	Begin Lit CC (→ 100) Ecrire CC, CC+10 → bloque Ecrire CC, CC+20	Begin Select count(*) from Voit where couleur="rouge" Select count(*) from Voit where couleur="rouge" Commit	 Begin Insert into Voit values (R4, "rouge") Commit

Tolérance aux pannes

- **Objectif: Garantir les propriétés d'Atomicité et de Durabilité quel que soit le type de pannes.**
- **Types de pannes:**
  - **Transaction failure (ex: deadlock, violation de CI)**
  - **System failure(ex: panne mémoire)**
  - **Media failure (panne disque)**
  - **Communication failure (panne réseau)**

Atomicité

- **Toutes les mises à jour d'une transaction doivent être prises en compte ou bien aucune ne doit l'être**
- **2 techniques de base**
  - Mises à jour en place ==>

Règle du **Write Ahead Logging (WAL)** :

Toute mise à jour est précédée d'une écriture dans un journal d'images avant permettant d'invalider cette mise à jour

  - Mises à jour dans un espace de travail
    ex: mécanisme de shadow pages

Journal d'images avant

- **But**: pouvoir défaire les mises à jour effectuées à tort
- **Journalisation physique**

  { <Trid, n° page, image avant de la page, action > }
- **Journalisation physique différentielle**

  { <Trid, n° page, offset1, offset2, image avant de la chaîne offset1-offset2, action > }
- **Journalisation logique**

  { <Trid, opération inverse avec paramètres d'appels> }

  ex: insert_tuple(t)------> delete_tuple(t)

  (technique de compensation)

Durabilité

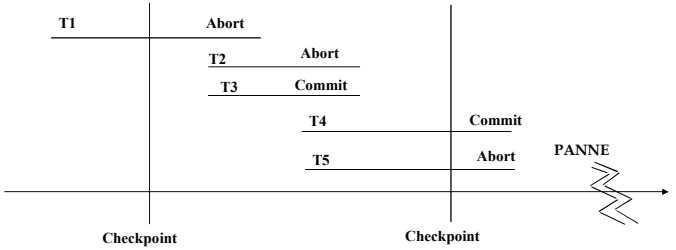
- **Objectif**: ne jamais perdre de mises à jour validées quel que soit le type de pannes
- **Moyen**: mémoriser toutes les mises à jour validées dans un journal d'images après. Ce journal a un format similaire au journal d'images avant
- **Règle du Force Log at Commit:**

  Le journal d'images après d'une transaction doit être écrit sur disque lors de la validation
- **Règle du bon sens:**

  Le journal d'images après doit être stocké sur un disque différent de la base de données

Checkpoint et fast commit

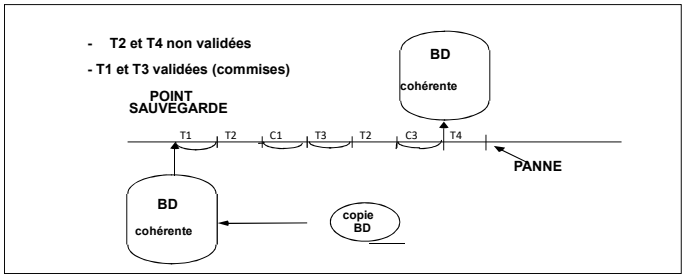
- **Objectif :** réduire le temps de latence pendant les validations et abandon de transactions afin de supporter un plus grand nombre de transactions par seconde (Tps)
- **Moyen** :
  - mises à jour du disque par une tâche asynchrone (implantation du STEAL/NO-FORCE)
  - seul le journal d'images après est écrit sur disque à la validation d'une transaction
  - complique toutefois les procédures de reprise



Reprise à froid

Procédure mise en oeuvre lors d'une *media failure*

- **RECHARGER** LA BASE DE DONNEES AVEC LA DERNIERE VERSION COHERENTE SAUVEGARDEE
- **REFAIRE** LES **TRANSACTIONS VALIDEES** ENTRE LE DERNIER POINT DE SAUVEGARDE ET LA PANNE, EN APPLIQUANT LE **JOURNAL DES IMAGES APRES**



Points de Sauvegarde

- **Introduction de points de sauvegarde intermédiaires**
- **Permet de ne défaire qu'une partie de la transaction en cas de problème**

Begin_Trans
- update
- update
- Savepoint (P1) // sauvegarde du contexte
- update
- update
- If condition Rollback to P1

Commit