



Université des sciences et de la Technologie Houari Boumediene
USTHB – Alger

Département d'Informatique

MASTER SYSTÈMES INFORMATIQUES INTELLIGENTS

MASTER INFORMATIQUE VISUELLE

MASTER ARCHITECTURES PARALLÈLES ET CALCUL INTENSIF

ARCHITECTURE ET ADMINISTRATION DES BASES DE DONNÉES

2016-2017

ENSEIGNANT : M. KAMEL BOUKHALFA

CONTRAINTES D'INTÉGRITÉ

GESTION DES ACCÈS CONCURRENTS

MÉCANISMES DE REPRISE

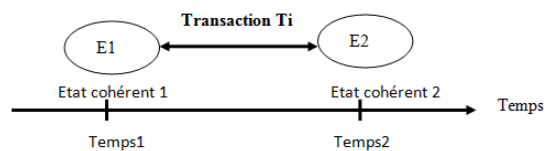
GESTION DES DROITS D'ACCÈS

GESTION DE L'INTÉGRITÉ

CONTRAINTES D'INTÉGRITÉ ET TRIGGERS

TRANSACTIONS ET GESTION DE TRANSACTIONS

- ❑ **Définition** : Une transaction est une unité de traitement séquentiel, exécutée pour le compte d'un usager, qui appliquée à une BD cohérente, restitue une BD cohérente.



- ❑ Une transaction est composée d'unités de traitement plus fines appelées Action
- ❑ Une action est une commande indivisible exécutée pour le compte d'une transaction par le système

EXEMPLE

Soit la table Employé

Employé (numemp, nomemp, prenomemp, fonction, salaire)

Transaction qui double le salaire d'un employé

READ (numemp=x, sal)

** / lire la donnée salaire pour le tuple dont x est la valeur de clé primaire*

*Nouv_salaire = sal*2*

*WRITE (numemp=x, nouv_salaire) * / écrire la donnée salaire pour le tuple dont x est la valeur de tuple*

PRIMITIVES DE GESTION DES TRANSACTIONS

- ❑ Une transaction est une séquence d'actions de lectures et d'écritures suivie d'actions de validation (Commit) et ou d'annulation (Abort) et ou de reprise (Rollback).
- ❑ **Validation** : exécution en fin de transaction d'une action spéciale « Commit » qui provoque l'intégration définitive de toutes les mises-à-jour de la transaction courante.
- ❑ **Annulation** : exécution d'une action « Abort » qui provoque l'annulation de toutes les mises-à-jour de la transaction courante.
- ❑ **Reprise** : exécution d'une action spéciale « Rollback » qui refait les mises-à-jour d'une transaction précédemment annulée.

PROPRIÉTÉS D'UNE TRANSACTION

- ❑ Une transaction est caractérisée par quatre propriétés dites ACID (Atomicité, Cohérence, Isolation, Durabilité).

- ❑ **Atomicité**

La séquence d'actions d'une transaction est indivisible. Si toutes les actions sont effectuées alors validation sinon annulation.

- ❑ **Cohérence**

L'exécution d'une transaction dans un environnement sans pannes préserve la cohérence de la BD. Le SGBD a la responsabilité de la cohérence de la BD en vérifiant le respect des contraintes d'intégrité.

PROPRIÉTÉS D'UNE TRANSACTION

- ❑ **Isolation**

Les actions d'une transaction sont isolées, c'est-à-dire que les transactions sont indépendantes les unes des autres. Les résultats intermédiaires ou partiels d'une transaction incomplète (la validation par le COMMIT non encore réalisée), (état temporairement incohérent) sont masqués aux autres transactions.

- ❑ **Durabilité**

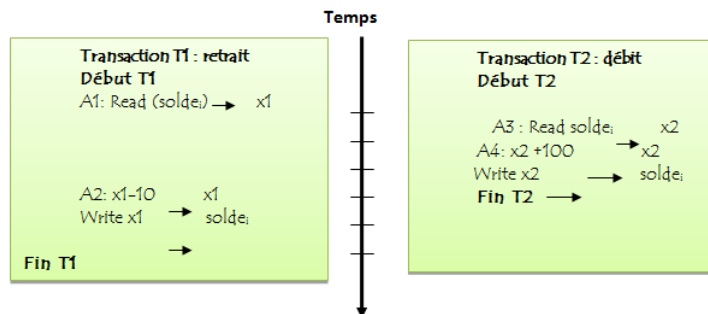
Les résultats d'une transaction complètement achevée (validée par un COMMIT) sont inscrites d'une manière durable dans la base de données. Il est impossible d'annuler les résultats d'une transaction qui s'est bien terminée.

LES PROBLÈMES DE CONCURRENCE

- ❑ Si les transactions sont parfaitement correctes prises chacune séparément, des problèmes peuvent survenir lorsqu'elles sont lancées en parallèle et que leur exécution nécessitent des entrelacements de leurs opérations respectives.
- ❑ Trois problèmes ont été répertoriés
 - ❑ La mise à jour perdue
 - ❑ La dépendance non validée
 - ❑ L'analyse incohérente

MISE À JOUR PERDUE

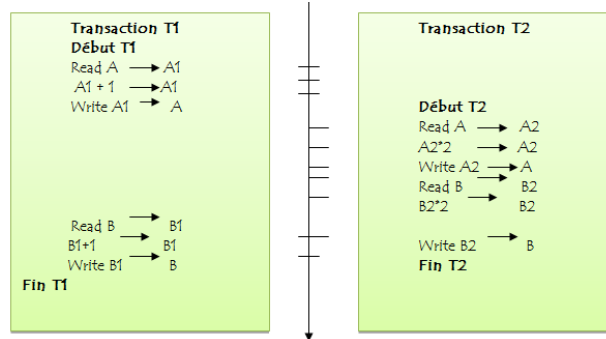
- ❑ Soit deux transactions T1 et T2 qui effectuent respectivement des débits et crédit sur le même compte i. A l'état initial le compte est à 10000 DA.



- ❑ A3 et A4 ont été effectuées entre A1 et A2, la M-A-J effectuée par A3 et A4 a été perdue.
- ❑ Il y a recouvrement de T2 par T1

DÉPENDANCE NON VALIDE

- ❑ La dépendance non validée ou lecture incorrecte, se produit lorsqu'une transaction est autorisée à voir les résultats d'une autre transaction avant que celle-ci ait été validée.
- ❑ Considérons une BD contenant les comptes A et B avec la CI : $A=B$



- ❑ Si Etat initial : $A = B = 10$
- ❑ Etat Final : $A = 22, B = 11$

PROBLÈME DE L'ANALYSE INCOHÉRENTE

- ❑ Il apparaît quand une transaction lit plusieurs valeurs de la BD, mais qu'une autre transaction en modifie certaines pendant l'exécution de la première transaction.



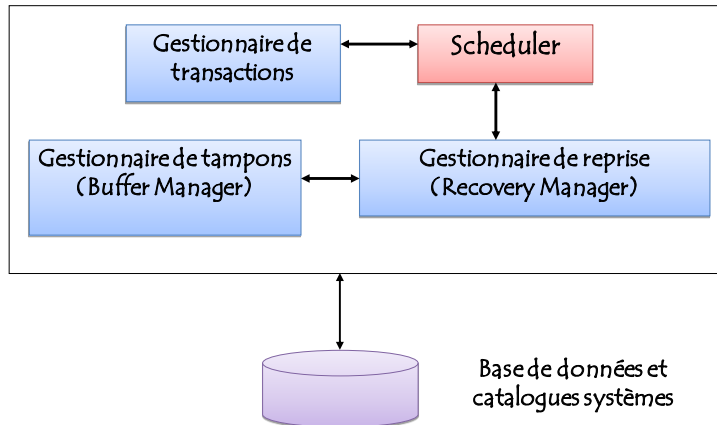
Explication : Etat initial : compte x = 100, compte y = 50, compte i = 25

T1 : transfert de 10 vers compte y donc x= 90 et y = 60

T2 : somme = 100+25+ 60 = 185 au lieu de 100+25+50= 175

RÉSOLUTION DES CONFLITS CONCURRENTS

- ❑ La résolution des conflits concurrents peut s'effectuer grâce à quatre modules de haut niveau qui gère les transactions



QUELQUES CONCEPTS

❑ Granule

- ❑ L'unité de base de données dont l'accès est contrôlé individuellement par le contrôleur. Cela peut être une page (ensemble de table), un fichier (une table), un enregistrement (un tuple).

❑ Une opération

- ❑ Une suite d'actions accomplissant une fonction sur un granule en respectant sa cohérence interne. Par exemple si le granule est une page, LIRE(page) et ECRIRE (page) sont les opérations de base et qui dans bien des systèmes également des actions (indivisibles).

❑ Un résultat d'une opération

- ❑ L'état du granule concerné après application de l'opération considérée.

L'ORDONNANCEMENT

- ❑ L'ordre dans lequel les opérations des transactions dans un environnement concurrent sont exécutées.
- ❑ **Ordonnement série** de deux transactions T1 et T2 consiste à exécuter en séquence toutes les opérations de T1, puis celles de T2 (pas de parallélisme).
- ❑ **Une exécution concurrente** sera correcte si elle produit les mêmes résultats et a les mêmes effets sur la BD qu'une exécution série de mêmes transactions.
- ❑ **Un ordonement série est correct.**

PRINCIPE DE SÉRIALISABILITÉ

- ❑ Critère permettant de dire que l'exécution d'un ensemble de transactions est correct.
- ❑ **L'exécution de transactions entremêlées est correcte** si elle est équivalente à une exécution des mêmes transactions en série (mêmes valeurs de départ et mêmes valeurs finales)
- ❑ **Un ordonement est sérialisable** s'il est correct et équivalent à un ordonement série formé des mêmes transactions
- ❑ **Un plan d'exécution de transactions (T1, T2, ..., Tn)** (Schedule) est une séquence d'actions obtenue en intercalant les diverses actions des transactions T1, T2, ..., Tn tout en respectant l'ordre interne des actions de chaque transaction.

OPÉRATIONS COMPATIBLES, PERMUTABLES

Opérations Compatibles

- Deux opérations O_i et O_j sont compatibles si et seulement si quelque soit l'exécution simultanée de O_i et O_j , le résultat de cette exécution est le même que celui de l'exécution séquentielle de O_i , O_j quelque soit l'ordre (O_i puis O_j ou O_j puis O_i)

```
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
}
```

O1, O2 sont compatibles

O1, O3 ne sont pas compatibles

OPÉRATIONS PERMUTABLES

Opérations Permutables

- Deux opérations O_i et O_j sont permutables si et seulement si toute exécution de O_i suivie de O_j donne le même résultat que celle de O_j suivi de O_i .
- Deux opérations travaillant sur 2 granules différents sont toujours compatibles et toujours permutables.
- O_1 , O_3 ne sont pas permutables
- O_1 , O_4 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
}
```

Théorème

- Une condition suffisante pour qu'une exécution soit sérialisable est qu'elle puisse être transformée par séparation des opérations compatibles et permutation des opérations permutables

OPÉRATIONS CONFLICTUELLES

- ❑ Deux opérations issues de deux transactions différentes sont en conflit si elles opèrent sur le même élément de données et au moins l'une d'entre elles est une action d'écriture.

$T_i : \text{READ}(X) \quad T_j : \text{WRITE}(X)$

$T_i : \text{WRITE}(X) \quad T_j : \text{WRITE}(X)$

$T_i : \text{WRITE}(X) \quad T_j : \text{READ}(X)$

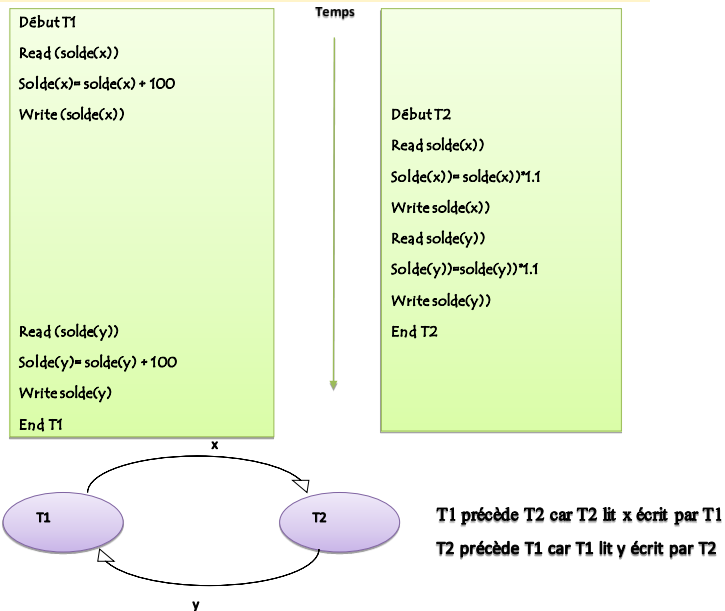
❑ Notion de précédence

Il y a précédence de T_i par T_j (T_i précède T_j , notée $T_i < T_j$) dans une exécution E si et seulement si, il existe deux opérations non permutables O_i et O_j telle que O_i est exécutée par T_i avant O_j par T_j .

GRAPHE DE PRÉCÉDENCE

- ❑ Un graphe de précédence est un graphe dirigé $G=(N,F)$ tel que : N ensemble de nœuds, F ensemble de flèches :
- ❑ créer un nœud pour chaque transaction
- ❑ créer une flèche dirigée $T_i \rightarrow T_j$ si T_j lit la valeur d'un élément écrit par T_i
- ❑ créer une flèche dirigée $T_i \rightarrow T_j$ si T_j écrit une valeur d'un élément après qu'il ait été lu par T_i
- ❑ créer une flèche dirigée $T_i \rightarrow T_j$ si T_j écrit une valeur d'un élément après qu'il ait été écrit par T_i
- ❑ Si une flèche existe de T_i vers T_j pour un ordonnancement O , alors dans tout ordonnancement sérialisable de O , T_i doit apparaître avant T_j .
- ❑ Si le graphe de précédence contient un cycle, l'ordonnancement n'est pas sérialisable.

EXEMPLE



MÉTHODES DE SÉRIALISATION

1. Sérialisation par estampillage
2. Sérialisation par Verrouillage

L'ESTAMPILLAGE

❑ Concept d'estampille

- ❑ Une estampille est une valeur numérique unique créée par le SGBD indiquant l'heure de démarrage relative d'une transaction
- ❑ Pour chaque transaction on lui associe une estampille. Deux manières possibles
 - ❑ L'horloge système
 - ❑ Utiliser un compteur.

❑ Protocole d'estampillage

- ❑ un protocole de contrôle de concurrence qui classe les transactions dans un ordre tel que les transactions les plus anciennes (qui portent l'estampille la plus petite) obtiennent la plus grande priorité en cas de conflits.

ALGORITHME BASÉ SUR LES ESTAMPILLES

- ❑ L'algorithme consiste à vérifier que les accès aux données par les transactions s'effectuent bien dans l'ordre affecté au lancement repéré par les estampilles.
- ❑ Principe : si une transaction tente de lire ou d'écrire une donnée, alors la lecture ou l'écriture n'est autorisée que si la dernière mise à jour de la donnée a été effectuée par une transaction plus ancienne.
- ❑ Sinon la transaction qui a demandée la lecture ou l'écriture est reprise avec une nouvelle estampille. (pour éviter qu'elles soient continuellement annulées ou redémarrées)

Si T_i émet un Read (A)

***Si $E(A) \leq i$ alors exécution de la lecture et $E(A) := i$
Sinon, l'action Read est rejetée, la transaction***

aussi.

Si T_i émet un Write (A)

***Si $E(A) \leq i$ alors exécution de l'écriture et $E(A) := i$
Sinon refuser l'écriture et rejeter la transaction.***

VERSION 2 DE L'ALGORITHME

- ❑ Amélioration de l'algorithme en utilisant deux estampilles pour la donnée : estampille de lecture (EL(A)) et estampille d'écriture (EE(A)).

La transaction T émet un Read (A)

si $E(T) < EE(A)$: {une transaction plus ancienne essaie de lire une donnée qui a été mise à jour par une transaction plus récente} alors l'action Read est rejetée et T est reprise.

Sinon, $E(T) \geq EE(A)$, exécution de Read (A) et $EL(A) = \max(E(T), EL(A))$

La transaction T effectue un Write (A)

Si $E(T) < EL(A)$: {une transaction plus récente utilise déjà la valeur courante de la donnée, ce qui serait une erreur de la modifier maintenant}, alors l'action Write est rejetée et reprise de la transaction.

Si $E(T) < EE(A)$: {la transaction T essaie d'écrire une valeur obsolète de la donnée}, alors rejeter l'action et reprise de la transaction.

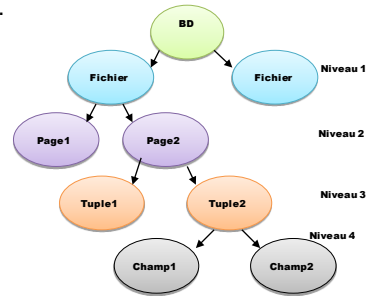
Sinon, $\{E(T) \geq EL(A) \text{ et } E(T) \geq EE(A)\}$, exécution de l'opération d'écriture et $EE(A) = E(T)$.

SÉRIALISATION PAR VERROUILLAGE

- ❑ Procédure employée pour contrôler les accès concurrents aux données en utilisant des verrous.
 - ❑ **Verrou partagé** : si une transaction dispose d'un verrou partagé SLOCK sur une donnée, elle peut lire la donnée mais pas la modifier
 - ❑ **Verrou exclusif** : si une transaction dispose d'un verrou exclusif XLOCK sur une donnée, elle peut lire et modifier cette donnée.
- ❑ A chaque ressource est donc associée
 - ❑ Une action de verrouillage qui peut être deux types :
 - ❑ **XLOCK(A) (verrou exclusif)** qui permet à une transaction d'acquies un contrôle exclusif de l'objet X
 - ❑ **SLOCK(A) (verrou partagé)** qui permet à une transaction de partager la lecture sur A. Ils sont aussi appelés verrou d'écriture (XLOCK) et verrou de lecture (SLOCK)
 - ❑ Une action de déverrouillage **UNLOCK(X)** qui permet la libération de l'objet X pour lequel on disposait d'un contrôle exclusif ou partagé.

GRANULARITÉ DES DONNÉES

- ❑ La granularité est la taille des données choisies comme l'unité de protection par un protocole de contrôle de concurrence.
- ❑ Elle peut être
 - ❑ La BD dans sa totalité
 - ❑ Un fichier
 - ❑ Une page
 - ❑ Un enregistrement (un tuple)
 - ❑ La valeur de champ d'un enregistrement



- ❑ Chaque fois qu'un nœud est verrouillé ses descendants le sont aussi.
- ❑ Si une transaction demande un verrou sur un granule d'un niveau i , alors les SGBD vérifie s'il n'existe pas de verrous sur les nœuds $i-1$ jusqu'à la racine.

PRINCIPE DE MANIPULATION DES VERROUS

- ❑ Toute transaction devant accéder à une donnée doit la verrouiller par **SLOCK** ou **XLOCK**
- ❑ Si la donnée n'est pas déjà verrouillée, celui-ci est accordé.
- ❑ Si la donnée est déjà verrouillée, le SGBD vérifie la compatibilité avec le verrou actuel
 - ❑ Si c'est un SLOCK que la transaction demande et que le verrou déjà placé est lui aussi un SLOCK, alors la demande peut être satisfaite
 - ❑ Sinon (verrou exclusif), la transaction doit attendre que le verrou se libère.
- ❑ Une transaction qui détient un verrou le conserve tant qu'elle ne le libère pas explicitement pendant l'exécution ou implicitement lorsqu'elle se termine.

EXEMPLE

<p>T1</p> <p>XLock (B)</p> <p>Read B</p> <p>B := B-50</p> <p>Write B</p> <p>Unlock(B)</p> <p>XLock (A)</p> <p>Read A</p> <p>A := A+50</p> <p>Write A</p> <p>Unlock A</p>	<p>T2</p> <p>SLock (A)</p> <p>Read A</p> <p>Unlock A</p> <p>SLock (B)</p> <p>Read B</p> <p>Unlock B</p> <p>Display (A+B)</p>
--	---

PROBLÈME DU VERROUILLAGE MORTEL (DEADLOCK)

- ❑ Le verrouillage mortel est l'impasse générée par deux (ou plusieurs) transactions qui sont en situation d'attente mutuelle que des verrous se libèrent.

<p>T1</p> <p>XLock (B)</p> <p>Read B</p> <p>B := B-50</p> <p>Write B</p> <p>XLock (A) : <i>T1 attend T2</i></p>	<p>T2</p> <p>XLock (A)</p> <p>Read A</p> <p>XLock (B) : <i>T2 attend T1</i></p>
---	--

PROTOCOLE DE VERROUILLAGE À DEUX PHASES

- ❑ Une transaction suit le protocole de verrouillage en deux phases si toutes les opérations de verrouillage **précèdent** la première opération de déverrouillage dans la transaction
- ❑ Chacune des transactions de la séquence émet des demandes de verrouillage et déverrouillage en deux phases :
 - ❑ **Verrouillage croissant** : une transaction peut obtenir des verrouillages mais pas de nouveau déverrouillage.
 - ❑ **Verrouillage décroissant (appelée aussi phase de résorption)**: une transaction peut obtenir des déverrouillages mais pas de nouveau verrouillage.

PROTOCOLE DE VERROUILLAGE À DEUX PHASES

- ❑ Si une transaction T veut lire (respectivement modifier) un objet, elle demande un *verrou partagé* (respectivement un *verrou exclusif*) sur l'objet.
- ❑ Une transaction peut relâcher un verrou avant de se terminer, mais elle ne peut plus redemander de verrou une fois qu'elle en a relâché un.

THÉORÈME DU V2P

- ❑ Si toutes les transactions satisfont le protocole de verrouillage à deux phases, tous les ordonnancements entrelacés sont sérialisables.
- ❑ Techniques de gestion des verrous indéfinis
 - ❑ Imposer un délai
 - ❑ Prévenir les verrous indéfinis
 - ❑ Les détecter et les récupérer.

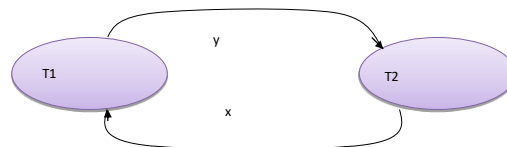
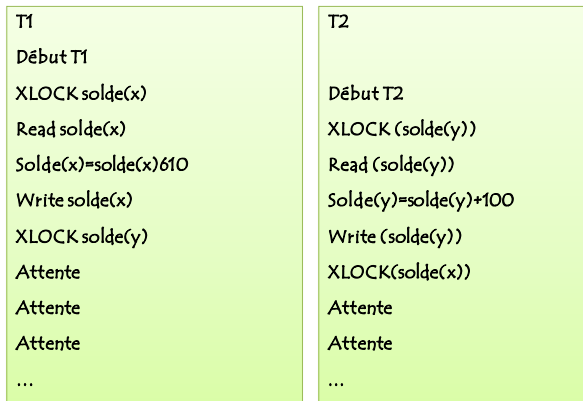
PRÉVENTION DES VERROUS INDÉFINIS

- ❑ **Délais impartis** : le SGBD impose des délais pour la libération de verrous. Si une transaction atteint ce délai, il est fort probable que soit la transaction est en deadlock, auquel cas, le SGBD va l'annuler puis va la reprendre.
- ❑ Prévention des verrous indéfinis :
 - ❑ **Algorithme Wait-Die (attendre-mourir)** : il permet à une transaction plus ancienne d'attendre une transaction plus récente, sinon la transaction est annulée (meurt) puis reprise avec la même estampille (elle devient la plus ancienne et ne meurt pas).
 - ❑ **Algorithme Wound-Wait (blessé attend)** : seule une transaction plus récente peut attendre une transaction plus ancienne.
 - ❑ **V2P conservateur** : au moment où la transaction démarre, elle demande tous ses verrous et ne démarre réellement que si elle les obtient tous. Ainsi les transactions ne sont jamais bloquées.

DÉTECTION DES VEROUS INDÉFINIS GRAPHE D'ATTENTE

- ❑ La détection des verrous indéfinis se fait par l'utilisation **des graphes d'attentes**
- ❑ Une transaction T_i dépend d'une transaction T_j quand la transaction T_j détient un verrou sur la donnée que T_i attend.
- ❑ Le graphe des attentes est un graphe dirigé $G = (N, F)$ constitué d'un ensemble de nœuds N et d'un ensemble de flèches F construit comme suit
 - ❑ Créer un nœud pour chaque transaction
 - ❑ Créer une flèche de T_i vers T_j quand la transaction T_i attend de verrouiller une donnée actuellement verrouillée par T_j .
- ❑ Un **verrou indéfini** existe si et seulement si le graphe des attentes contient un **cycle**.

EXEMPLE



REPRISE APRÈS **PANNES**