III- PROGRAMMATION TRANSACTIONNELLE ET NON-PROCEDURALE

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

1- CONCEPTS DE BASE

1.1- Notion de cohérence

- **CONTRAINTE D'INTÉGRITÉ : On appelle contrainte d'intégrité dans une base de données toute assertion qui doit être vérifiée par les données à des instants déterminées**

- **Exemple : Base de données du personnel d'une entreprise**

  - Contraintes de domaine : "Tout employé doit avoir au moins 18 ans"
  - Contraintes de mise à jour : "Tout salaire ne peut être diminué"
  - Les dépendances : - fonctionnelles (clé primaire)
  - multivalués
  - d'inclusion (clé étrangère)
  - Contraintes arithmétiques, temporelle ... : Sol = Crédit - Débit

- **BASE DE DONNÉES COHÉRENTE : Toute base de données dont l'ensemble des contraintes d'intégrité (explicites ou implicites) est respecté par les données de la base est dit cohérente ou dans l'état cohérent**

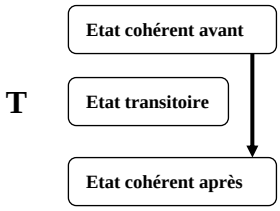
III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

1- CONCEPTS DE BASE
1.2- NOTION DE TRANSACTION

-> Une transaction est une unité logique atomique de traitement ayant 3 propriétés suivantes :

- Atomicité : une transaction est soit complètement exécutée soit complètement abandonnée
- Cohérence: une transaction fait passer la base de données d'un état cohérent à un autre état cohérent Si une transaction ne va pas à son terme pour une raison ou pour une autre la base de données est restaurée dans l'état où elle se trouvait avant que la transaction ne démarre
- Sérialité : une exécution en // n transactions doit donner un même résultat qu 'une exécution séquentielle de ces n transactions

-> Exemple
Ecriture comptable à double : Transaction T crédite C2 et débite C1



(1) début transaction
(2) lire C1
(3) C1 <-- C1 - X
(4) écrire C1
(5) lire C2
(6) C2 <-- C2 + X
(7) écrire C2
(8) fin transaction

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

1- CONCEPTS DE BASE
1.3- Vie d'une transaction

- **VIE SANS HISTOIRE**
  Une transaction s'exécute normalement jusqu'à la fin. Elle se termine par une instruction de validation de tous les effets qu'elle a produit sur la base de données.On dira que cette transaction est validée : toutes les modifications sur la base de données permettent la constitution de la nouvel état de cohérence de la base

- **UN ASSINSSINAT**
  Un évènement extérieur vient interrompre l'exécution de la transaction de façon irrémédiable. Cet arrêt de la vie d'une transaction peut provenir soit d'une panne soit d'une action délibérée de la part du SGBD qui décide de supprimer telle ou telle transaction

- **UN SUICIDE**
  Au cours de son exécution, la transaction détecte certaines conditions qui font que la poursuite de son exécution s'avère impossible. Elle peut, dans ce cas, décider de se supprimer en s 'exécutant une instruction d'annulation

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

1- CONCEPTS DE BASE

1.4- Problèmes de concurrence d'accès

- **EXEMPLE ILLUSTRE**

  Soient T1 et T2 deux transactions qui s'intéressent à un même objet A. Etant donnée "lire" et écrire deux seules opérations possibles sur A, nous avons 4 cas possibles

  - **Lecture-Lecture : Aucune conflit : un même objet peut toujours être partagée en lecture**
  - **Ecriture-Ecriture : Ce cas peut induire des pertes de mises à jour :**

  T2 vient "écraser" par une autre écriture celle effectuée par T1

  sans tenir compte cette dernière

temps	T1	T2	A
t1	lire A	-	A = 10
t2	-	lire A	
t3	A:= A+10	-	
t4	-	-	
t5	-	A:= A+50	
t6	écrire A	-	A = 20
t7	-	écrire A	A = 60

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

1- CONCEPTS DE BASE

1.4- Problèmes de concurrence d'accès

- **EXEMPLE ILLUSTRE**

  • Ecriture-Lecture : lectures impropre

  T2 lit une valeur modifiée par T1 et ensuite T1 est annulée. On a une valeur "impropre" de A dans T2

temps	T1	T2	A
t1	lire A	-	A = 10
t2	A:=A+20	-	
t3	écrire A	-	A = 30
t4	-	lire A	
t5	Annulation	-	A = 10
t6	-	-	

  • Lecture-Ecriture : lectures non reproductives : T1 modifie la valeur de A entre deux lectures de T2

temps	T1	T2	A
t1	lire A	-	A = 10
t2	A:=A+20	lire A	
t3	A:=A+10	-	
t4	écrire A	-	A=30
t5	-	-	
t6	-	lire A	

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

2- EXÉCUTIONS SANS CONFLIT ET SÉRIALISABLES

2.1- Concepts de base

- **CONTRÔLEUR (SCHEDULER) : Un contrôleur transactionnel est un module système chargé de contrôler les accès concurrent aux données**
- **GRANULE : On appelle granule l'unité de base de données dont l'accès est contrôlé individuellement par un contrôleur**
- **OPÉRATION : Une opération est une suite d'actions élémentaires accomplissant une fonction sur un granule en respectant sa cohérence interne**
- **RÉSULTAT DE L'OPÉRATION : Le résultat d'une opération est l'état du granule concerné après l'application de l'opération considérée anssi que les effets de bords provoqués par l'opération**
- **EXÉCUTION DE TRANSACTIONS (SCHEDULE - LOG) : Une exécution des transactions (T1, T2, ..., Tn) est une séquence d'actions élémentaires obtenues en interclassant les diverses actions des transactions concernant**

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

2- EXÉCUTIONS SANS CONFLIT ET SÉRIALISABLES

2.1- Concepts de base

- **Exemple**

T1	T2
Lire A → a1	Lire A → a2
a1 + 1 → a1	a2 * 2 → a2
Ecrire a1 → A	Ecrire a2 → A
Lire B → b1	Lire B → b2
b1 + 1 → b1	b1 * 2 → b2
Ecrire b1 → B	Ecrire b2 → B

EXÉCUTION 1	EXÉCUTION 2
T1: Lire A → a1	T2: Lire A → a2
T1: a1 + 1 → a1	T2: a2 * 2 → a2
T1: Ecrire a1 → A	T1: Lire A → a1
T2: Lire A → a2	T1: a1 + 1 → a1
T2: a2 * 2 → a2	T2: Ecrire a2 → A
T2: Ecrire a2 → A	T2: Lire B → b2
T1: Lire B → b1	T2: b1 * 2 → b2
T1: b1 + 1 → b1	T1: Ecrire a1 → A
T1: Ecrire b1 → B	T1: Lire B → b1
T2: Lire B → b2	T1: b1 + 1 → b1
T2: b1 * 2 → b2	T1: Ecrire b1 → B
T2: Ecrire b2 → B	T2: Ecrire b2 → B

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

2- EXÉCUTIONS SANS CONFLIT ET SÉRIALISABLES
2.2- Exécution sérialisable (sérialité)

- **SUCCESSION (SERIAL SCHEDULE)**

Une exécution E de T1, .., Tn est une succession s'il existe une permutation š de (1, 2, ..,n) telle que

E = < Tš(1), Tš(2), .., Tš(n) >

- **PROPRIÉTÉ**

Une succession est une exécution sans perte d'opérations ni inconsistance

- **EXÉCUTION SÉRIALISABLE (SERIALIZABLE SCHEDULE)**

Une exécution quelconque de T1, .., Tn est dit sérialisable (ou sérialité) si et seulement si elle donne pour chaque transaction participante le même résultat qu'une succession de T1, .., Tn

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

2- EXÉCUTIONS SANS CONFLIT ET SÉRIALISABLES
2.3- Propriétés des opérations

- **OPÉRATIONS COMPATIBLES : Deux opérations O1, O2 sont compatibles si et seulement si quel que soit l'exécution simultanée de O1 et O2, le résultat de cette exécution est le même que cemui de l'exécution séquentielle de O1 suivie de O2 ou de O2 suivie O1**

- **PROPRIÉTÉ : Deux opérations travaillant sur deux granules différents sont toujours compatibles**
- **Note : En général, le résultat de <O1, O2> peut être différent de celui de <O2, O1>**

- **OPÉRATIONS PERMUTABLES : Deux opérations O1, O2 sont permutables si et seulement si toute exécution de O1 suivie O2 donne le même résultat que celle de O2 suivie O1**

- **PROPRIÉTÉ**

  Tout couple d'opérations compatible est permutable

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

2- EXÉCUTIONS SANS CONFLIT ET SÉRIALISABLES

2.3- Propriétés des opérations

- **EXEMPLE :**

O11- Lire A → a1 **a1 + 1 → a1** **Print a1**	**O21- Lire A → a2** **a2 * 2 → a2** **Ecrire a2 → A**
O12- Lire A → a1 **a1 + 1 → a1** **Ecrire a1 → A**	**O22- Lire A → a2** **a2 * 2 → a2** **Ecrire a2 → A**
O13- Lire A → a1 **a1 + 1 → a1** **Ecrire a1 → A**	**O23- Lire A → a2** **a2 + 10 → a2** **Ecrire a2 → A**

- O11 et O21 sont compatibles

- O13 et O23 sont permutables

- O12 et O22 ne sont pas compatibles ni permutables

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

2- EXÉCUTIONS SANS CONFLIT ET SÉRIALISABLES

2.4- Caractéristiques des exécutions sérialisables

- **TRANSFORMATIONS DE BASE**

- **Séparation d'opérations : La séparation d'opérations consiste à isoler des couples des opérations compatibles dans une exécution et de remplacer chaque couple E(Oi, Oj) par la séquence donnant le même résultat : soit <Oi, Oj> soit <Oj, Oi>**
- **Permutation d'opérations : La permutation d'opérations consiste à isoler les couples des opérations permutables et de changer l'ordre d'exécution des opérations dans un même couple**

- **PROPRIÉTÉ**

  Les transformations de base : sélection d'opérations et permution d'opérations conservent le résultat d'une exécution

- **1e CONDITION SUFFISANTE (de l'exécution sérialisable)**

  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 en une succession des transactions composantes

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

2- EXÉCUTIONS SANS CONFLIT ET SÉRIALISABLES
2.4- Caractéristiques des exécutions sérialisables

- **EXEMPLE**

O2 et O3 sont permutables. On obtient donc <T1, T2> :

O1:	**O1:**
T1: Lire A → a1	**T1: Lire A → a1**
T1: a1 + 1 → a1	**T1: a1 + 1 → a1**
T1: Ecrire a1 → A	**T1: Ecrire a1 → A**
O2:	**O3:**
T2: Lire A → a2	**T1: Lire B → b1**
T2: a2 * 2 → a2	**T2: b1 * 2 → b2**
T2: Ecrire a2 → A	**T1: Ecrire b1 → B**
O3:	**O2:**
T1: Lire B → b1	**T2: Lire A → a2**
T1: b1 + 1 → b1	**T2: a2 * 2 → a2**
T1: Ecrire b1 → B	**T2: Ecrire a2 → A**
O4:	**O4:**
T2: Lire B → b2	**T2: Lire B → b2**
T2: b1 * 2 → b2	**T2: b1 * 2 → b2**
T2: Ecrire b2 → B	**T2: Ecrire b2 → B**

L'exécution précédente est une exécution sérialisable

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

2- EXÉCUTIONS SANS CONFLIT ET SÉRIALISABLES
2.4- Caractéristiques des exécutions sérialisables

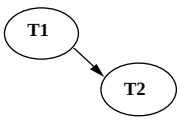
- **GRAPHE DE PRÉCÉDENCE : Soit S une exécution sans opération simultanée. On dit que Ti précéde Tj dans S, et noté Ti < Tj, si et seulement s'il existe deux opérations non permutables Oi et Oj telles que Oi est exécutée par Ti et Oj est exécutée par Tj**

- **2e CONDITION SUFFISANTES (de l'exécution sérialisable)**

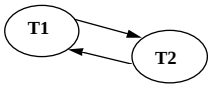
  Une condition suffisante pour qu'une exécution soit sérialisable est que le graphe de précédence associé soit sans circuit

- **EXEMPLE**

Sérialisable :
T1: a1 + 1 Æ a1
T2: a2 * 2 Æ a2
T1: b1 + 1 Æ b1
T2: b1 * 2 Æ b2



Non sérialisable :
T1: a1 + 1 Æ a1
T2: a2 * 2 Æ a2
T2: b1 * 2 Æ b2
T1: b1 + 1 Æ b1



III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.1- Principes de base

- **STRATÉGIES**
- **Les stratégies de verrouillage consistent à éviter la génération d'exécution incorrecte en faisant attendre les transactions voulant exécuter des opérations conflictuelles sur le même granule**
- **Le premier objectif des algorithmes de verrouillage es de ne laisser s'exécuter simultanément ques des opérations compatibles**
- **MODE D'OPÉRATION**
- **Les modes d'opération sont des caractéristiques permettant de classer une opération et de déterminer ses compatibilités avec les autres opérations**
- **Les modes simples : m0=LIRE 1 0 m1=ECRIRE 0 0**
- **Les modes définis par le groupe DBTG CODASYL :**
  - **M1 = consultation non protégée 1 1 1 1 0 0**
  - **M2 = consultation protégée 1 1 0 0 0 0**
  - **M3 = mise à jour non protégée 1 0 1 0 0 0**
  - **M4 = mise à jour protégée 1 0 0 0 0 0**
  - **M5 = consultation exclusive 0 0 0 0 0 0**
  - **M6 = mise à jour exclusive 0 0 0 0 0 0**

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.2- Protocole de verrouillage

- **DESCRIPTION**

- **Un protocole de verrouillage est un protocole d'accès à des granules partagés caractérisé par des demandes d'autorisation d'opérations et de signalements de fin d'opérations.**
- **Un protocole de verrouillage, en général, se compose de deux actions spéciales :**

  - LOCK (g, M) permet à une transaction d'indiquer au contrôleur le début d'une opération correspondant au mode M sur le granule g

  - UNLOCK (g) permet à une transaction d'indiquer au contrôleur la fin d'une opération encours sur le granule g
- **Le protocole de verrouillage nécessite donc l'exécution de deux actions supplémentaires LOCK et UNLOCK. Ce deux actions peuvent être automatiquement ajoutées par le contrôleur, chaque fois qu'il exécute une opération. Elles sont donc cachées aux utilisateurs finaux**
- **La plupart de système effectue le déverrouillage soit à la fin de la transaction soit en des points intermédiaires où la base de données se trouve à l'état cohérent. Cela permet la mise en oeuvre des mécanismes de reprise en cas de pannes**

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.3- Algorithmes de verrouillage

- **PRINCIPES**

- **Mémoriser les modes d'opérations en cours sur chaque granule (mettre des verrous)**

- **Pour cela, on associe, à chaque couple granule g et transaction Ti opérant sur ce granule, un vecteur de bits dont chacun représente le verrou d'un mode d'opération possible sur g par Ti**

$$A(g,i) = \begin{bmatrix} a1 \\ a2 \\ . \\ ak \end{bmatrix}$$

où : aj = 1 si le mode Mj est en cours d'exécution pour le compte de la transaction i sur le granule g, et 0 sinon

- **De manière similaire, on peut présenter le mode M demandé lors de l'exécution d'une action LOCK (g,M) sur le granule g, par un vecteur de bits :**

$$M = \begin{bmatrix} m1 \\ m2 \\ . \\ mk \end{bmatrix}$$

où : mj = 1 si le mode Mj est demandé, et 0 sinon

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.3- Algorithmes de verrouillage

- **THÉOREME (de compatibilité)**

  Les modes d'opérations demandées lors d'une primitive LOCK (g, M) exécutée par une transaction Ti sont compatibles avec les modes en cours d'exécution sur g par les autres transactions si et seulement si :

$$M \subset \neg (\neg C * \sum_{k \neq i} A(g,k))$$

où $\sum$ **est la somme logique des vecteurs booléens**

$\neg$ **est la négation des matrices booléens**

C est la matrice de compatibilité

*** est le produit matriciel booléen (l'élément (i,j) de la matrice de produit est la somme booléen des multiplications des éléments de la lignes i^e de la matrice gauche et de j^e colonne de la matrice droite)**

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.3- Algorithmes de verrouillage

- **THÉOREME (de compatibilité)**

Preuve

- $\sum$ A(g,k) est un vecteur de bits dont le bit j est à 1 si et seulement si le mode Mj est en cours d'exécution sur g par la transaction k autre que Ti. ¬C est le complément de la matrice compatibilité.

- On a donc:

¬C * $\sum$ A(g,k) est un vecteur de bits dont le bit j est à 1 si et seulement si le mode Mj est incompatible avec un mode en cours d'exécution sur le granule g par une transaction autre que Ti.

- Finalement, ¬(¬C * $\sum$ A(g,k)) est donc un vecteur de bits représentant tous les modes d'opérations compatibles avec les modes en cours d'exécution sur le granule g par une transaction autre que Ti. Tout vecteur inclus dans ce dernier correspond donc à des modes compatibles avec ceux en cours d'exécution sur g par une transaction autre que Ti. (CQFD)

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.3- Algorithmes de verrouillage

- **ALGORITHME :**

Supposons que Q[g] est la file d'attente associée au granule g. Cette fille d'attente contient les demandes de verrouillage d'une transaction k en mode M : (k, M). On peut donc établir les algorithmes suivants :

```
Procédure LOCK (g, M) ;
    si      M ⊂ ¬ ( ¬C * ∑ A(g,k))
                        k ≠ i
    alors A(g,i) := A(g,i) + M ;
    sinon  " insérer (i,M) dans Q[g] " ;
           " bloquer la transaction Ti ";
    finsi;                          1
fin LOCK;
```

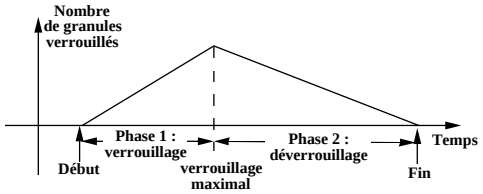
```
Procédure UNLOCK (g) ;
    A(g, i) := (0) ;
    Pour chaque (q,M') de Q[g] faire
       si  M' ⊂ ¬ ( ¬C * ∑ A(g,k))
                        k ≠ q
       alors  A(g,q) := A(g,q) + M ;
              "extraire (q, M') de Q[g]";
               "débloquer la transaction Tq";
       finsi;
    fin pour;
 fin UNLOCK;
```

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.4- Algorithmes de verrouillage à deux phases

- **RESTRICTION DEUX PHASE : On appelle transaction deux-phases, toute transaction qui n'exécute pas de LOCK après avoir exécuté UNLOCK**



- **THÉORÈME : DE SÉRIALISABILITE**

  Toute exécution complète d'un ensemble de transaction deux-phases est sérialisable

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.4- Algorithmes de verrouillage à deux phases

- **Preuve : (par l'absurde)**
- **Une exécution sérialisable est une exécution dont le graphe de précédence ne possède pas de circuits.**
- **Considérons alors une exécution complète d'un ensemble :**

  [T1, T2, .., Tn] de transactions deux phases et supposons qu'il existe un circuit de précédence : Ti1 → Ti2,...→..., Tin → Ti1
- **Il découle immédiatement que :**

  - Ti2 "LOCK" un granule gi1 après que Ti1 "UNLOCK" ce granule gi1,

  - Ti3 "LOCK" un granule gi2 après que Ti2 "UNLOCK" ce granule gi2,

  - ...

  - Ti1 "LOCK" un granule gin après que Tin "UNLOCK" ce granule gin.
- **Chaque transaction Ti2, Ti3, ... étant deux-phase, elle exécute "UNLOCK" seulement lorsqu'elle a terminé tous ses "LOCK".**

  - De ce fait, Ti1 exécute "UNLOCK" sur gi1 avant d'exécuter "LOCK" sur gin. Par définition elle n'est donc pas une transaction deux phases. Ce qui est contraire aux hypothèses.

(CQFD)

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.4- Algorithmes de verrouillage à deux phases

- **RÈGLES D'UTILISATION DE "LOCK/UNLOCK"**

R1- Toute transaction doit exécuter LOCK avec le(s) mode(e) d'opération correct(s) et le granule choisi avant d'exécuter(une opération sur ce granule;

R2- Toute transaction doit exécuter UNLOCK sur le granule choisi plus ou moins longtemps après l'exécution de l'opération;

R3- Toute transaction ne peut exécuter LOCK après avoir exécuté UNLOCK

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

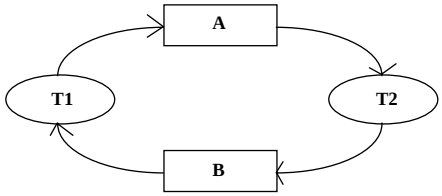
3.5- Problème de verrou mortel (interblocage)

- **INTERBLOCKAGE (Deadlock)**

  Le verrou mortel ou interblocage est la situation à laquelle on aboutit lorsque des granules ont été verrouillés dans un ordre tel qu'un groupe de transactions vérifie les deux propriétés suivantes :

  1- Chaque transaction du groupe est bloquée en attente d'un granule;

  2- L'exécution supposée de toutes les transactions n'appartenant pas au groupe ne permet pas de débloquer une des transactions du groupe



SOLUTIONS : la détection, la prévention

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

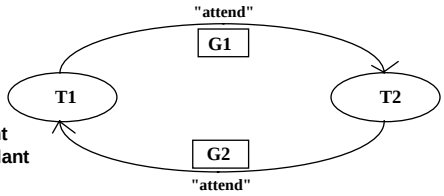
3.5- Problème de verrou mortel (interblocage)

- **GRAPHE DES ATTENTES (Murphy 68)**

  Le graphe des attentes est un graphe G(T, G) où

  - T est l'ensemble de transactions concurrentes [T1, T2, .., Tn] se partageant les granules G1, G2, .., Gm, et

  - G est la relation "attend" définie par : Tp "attend" Tq si et seulement si Tp attend le verrouillage d'un granule gi qui est déjà verrouillé par Tq



- **THÉORÈME (Murphy 68)**
  Il existe une situation d'interblocage dans une exécution E si et seulement si le graphe des attentes correspondant à E possède un circuit

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.5- Problème de verrou mortel (interblocage)

- **GRAPHE DES ALLOCATIONS (HOLT 72)**

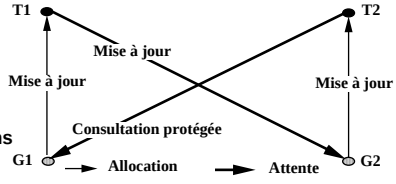
  Le graphe des allocations est un graphe qui se compose de deux ensembles de sommets :

  - l'ensemble de transactions concurrentes [T1, T2, .., Tn] et

  - l'ensemble de granules G1, G2, .., Gm partagés.

- **Un arc relie le granule Gi à la transaction Tp si et seulement si Tp a obtenu un verrouillage sur Gi dans un mode d'opération; l'arc est valué par les modes d'opérations alloués.**
- **Un arc relie la transaction Tp au granule Gi si et seulement si Tp a demandé et n'a pas encore obtenu l'allocation de ce granule; l'arc est étiqueté par les modes d'opérations demandé**

- **THÉORÈME DE VERROU MORTEL (HOLT 72)**
  Une condition nécessaire d'existence d'interblocage est la présence d'un circuit sur le graphe des allocations

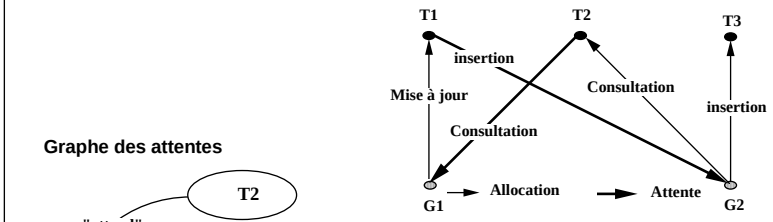


III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

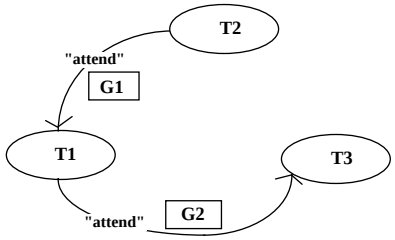
3.5- Problème de verrou mortel (interblocage)

- **EXEMPLE : graphe d'allocation avec circuit mais pas de verrou mortel**



Graphe d'allocation

Graphe des attentes



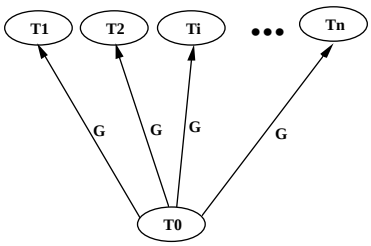
III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.6- Autres problèmes

- **PROBLÈME DE FAMINE (BLOCAGE PERMANENT)**

  Le problème de famine existe quand un groupe de transactions se coalise, en effectuant des opérations compatibles entre elles, contre une transaction individuelle qui désire effectuer une opération incompatible avec les précédentes. La transaction individuelle peut alors attendre indéfiniment



III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.6- Autres problèmes

- **PROBLÈME DES FANTÔMES (Eswaran 76)**

  Le problème des fantômes survient lorsqu'une entité est introduite dans la base de données et ne peut être traitée par une transaction en cours qui devrait logiquement la traiter

	Passager			Vol
Nom	n° vol	n° siège	n° vol	nb passagers
Durant	100	10	100	4
Martin	100	5		
Durand	100	3		
Satre	100	14		

T1 : (1e partie) : lister la relation Passager (T1a)
T1 : (1e partie) : lister la relation Vol (T1b)
T2 : inserer dans Passager le tuple (Satre, 100, 14) et incrémenter le nombre de passagers du vol n° 100
- Hypothèse : Les transactions s'exécutent dans l'ordre E = (T1a, T2, T1b)

- E est une exécution valable mais le résultat de T1 est une liste de 3 nom alors que le nombre de passagers est 4. Satre est un "fantôme" dans cette exemple

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.7- Prévention des interblocages

- **PRINCIPE : La prévention des interblocages consiste à supprimer l'une des conditions qui rend possible les situation de verrou mortel. On a de classes des techniques : préordonnancement des ressources (difficile à mettre en oeuvre) et préordonnancement des transactions**

- **METHODE DIE-WAIT : Quand une transaction Ti demande à verrouiller un granule qui est déjà verrouillé par une transaction Tj dans un mode incompatible, Ti attend Tj si et seulement si i < j (une transaction plus vieille attend une plus jeune). Dans le cas contraire Ti se suicide et elle sera reprise avec le même estampille (la jeune se suicide et la plus vielle ne mourras pas jamais)**

- **METHODE WOUND-WAIT : Quand une transaction Ti demande à verrouiller un granule qui est déjà verrouillé par une transaction Tj dans un mode incompatible, Ti attend Tj si et seulement si i > j (une transaction plus jeune attend une plus vielle). Dans le cas contraire Ti tue le Tj et elle sera reprise avec le même estampille (une plus vielle tue une plus jeune)**

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

4.8- Détection des interblocages

- **PRINCIPE : Un algorithme de détection de l'interblocage peut se déduire d'un algorithme de détection de circuits appliqué au graphe des attentes (ou des allocations). Soit N(k) le nombre de granules dont la transaction Tk attend le verrouillage :**

  1) sur le graphe des attentes G, un sommet est pendant si la transaction qu'il représente n'attend le verrouillage d'aucun granule. On peut réduire le graphe G en supprimer les sommets pendants (N(k) = 0)

  2) recalculer les N(k) sur le graphe G réduit : en comptant les demandes qui peuvent être satisfaites après la réduction et en décrémentant N(k) chaque fois que l'on compte une demande de transaction Tk.

  3) revenir à 1) si les N(k) sont changés et G non vide, sinon 4)

  4) Si G est vide alors : Pas de circuits
  Sinon Circuit est détecté

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

3- CONTRÔLE D'EXÉCUTION SÉRIALISABLE : ALGORITHMES DE VERROUILLAGE À DEUX PHASES

3.8- Détection des interblocages

- **ALGORITHME**

```
Booléenne Procedure SLOCK (gi, Q, K);
    "retourner VRAI si la demande Q de la transaction Tk, sur le granule gi, peut-être
    satisfaites compte tenu de l'état d'allocation des granules aux transactions présentes
    dans le graphe des attentes, et FAUX dans autres cas"
Fin SLOCK;
Booléenne Procédure DÉTECTER
    T = {liste des transactions Tj telles que N(j) ° 0 } ;
    R = {liste des granules verrouillés par les transactions de T} ;
    Pour "chaque entrée gi de R" faire
            Pour "chaque demande Q de Tk en attente de gi et non marquée" faire
              Si    SLOCK(gi, Q, k) = VRAI alors  "Marquer Q"; N(k) = N(k) - 1;
                    Si N(k) = 0 alors "Sortir Tk de T" ;
                                      "Ajouter le granules verrouillés par Tk à la liste R";
    Fin si;  Fin si;  Fin Pour; Fin Pour;
    Si T = Ø Alors    DETECTER = FAUX;  Sinon DETECTER = VRAI;  Fin si;
Fin DETECTER;
```

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

4- MODELES DE CONTRÔLE D'EXÉCUTION TRANSATIONNEL
4.1- MODELE LINEAIRE

- **PRINCIPE :**

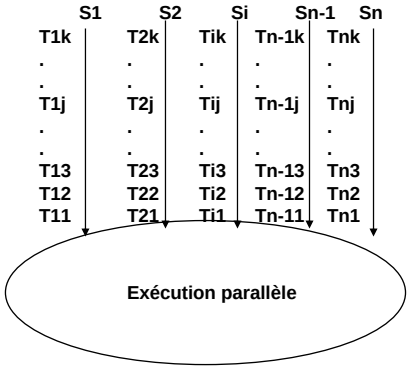
- S1, S2, Sn sont n sessions de travail indépendantes

- Dans chaque session, les transactions sont exécutées séquentiellement

- Entre les sessions, les transactions sont exécutées en parallèle

COMMIT

ROLLBACK



III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

4- MODELES DE CONTRÔLE D'EXÉCUTION TRANSATIONNEL
4.2- MODELE HIERARCHIQUE

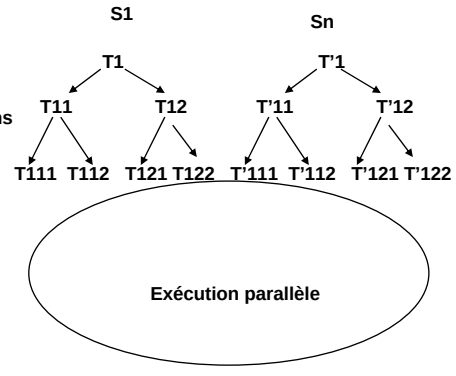
- **PRINCIPE :**

- S1, S2, Sn sont n sessions de travail indépendantes

- Dans chaque session, les transactions sont imbriquées dans une hiérarchie

- Entre les sessions, les transactions sont exécutées en parallèle

BEGINT
ENDT
COMMIT
ROLLBACK



III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

5- PROGRAMMATION NON PROCEDURALE ET TRANSATIONNELLE : SQL
5.1-PRINCIPE

*** Pilotés par les données**

*** Non procédural (QUOI mais non COMMENT)**

*** Plusieurs enrégistrements (tuples) manipulés par chaque commande**

*** La puissance au moins équivalente à celle de l'algèbre relationnelle**

*** Adaptation pour chaque type d'utilisateur**

- **adminstrateurs,**
- **utilisateurs finaux,**
- **programmeurs d'application**

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

5- PROGRAMMATION NON PROCEDURALE ET TRANSATIONNELLE : SQL
5.2-HISTORIQUE

SQL = Structured Query Language

*** Origine : IBM**
travaux de Codd et de groupe du System-R au milieu des années 70 ----- SEQUEL

*** Adopté et commercialisé par Oracle à la fin 70**

*** Imposé par IBM comme le standard (de facto) pour langages relationnels**

*** Devenu standard ANSI en Octobre 1989**
- **norme X3.135 spécifie 2 niveaux**

III- PROGRAMMATION TRANSACTIONNELLE ET NON PROCEDURALE

5- PROGRAMMATION NON PROCEDURALE ET TRANSATIONNELLE : SQL

5.3- GROUPES DE COMMANDES

*** Commandes de définition des données**

CREATE	**Créer des tables, des vues et index**
ALTER	**Modifier des tables**
DROP	**Supprimer des tables, vues et index**

*** Commandes de manipulation des données**

INSERT	**Insérer des données**
UPDATE	**Mettre à jour des données**
DELETE	**Supprimer des données**
SELECT	**Recherche des données**

*** Commandes de contrôle des données**

GRANT	**Attribuer des droits d'accès aux tables**
REVOKE	**Retirer des droits d'accès aux tables**
COMMIT	**Exécuter (valider) une transaction**
ROLLBACK	**Abandonner une transaction**