SGBD : Bases de données avancées [M3106C]

Hocine ABIR

9 septembre 2014

IUT Villetaneuse
E-mail: abir@iutv.univ-paris13.fr

TABLE DES MATIÈRES

3 Gestion des Accès Concurrents (1) 1
3.1 Introduction 1
3.2 Concepts de base 1
3.3 Caractéristiques des ordonnancements acceptables 4
3.4 Ordonnancement par estampillage à l'initialisation 8
3.5 Ordonnancement par verrouillage à deux phases 11

CHAPITRE

TROIS

Gestion des Accès Concurrents (1)

3.1 Introduction

La gestion des accès concurrents dans un SGBD est la méthode utilisée par celui-ci pour permettre à plusieurs utilisateurs d'accéder simultanément aux mêmes données.

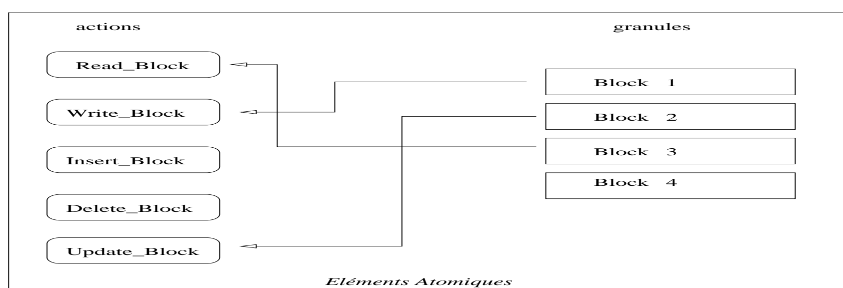
Cette méthode doit garantir :

1. cohérence de la base : lecture et mise à jour consistentes des données ,
2. partage des données : concurrence maximale des accès aux données,
3. temps de réponse : bonnes performances.

3.2 Concepts de base

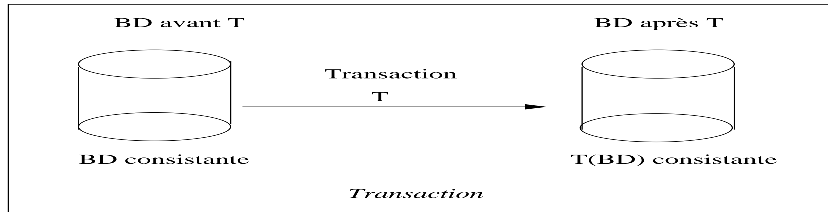
3.2.1 Deux élements atomiques

1. granule : ensemble de données constituant une unité de base (indivisible) d'accès : permet les "accès de base" de la concurrence.
2. action : suite d'opérations constituant une unité de base (indivisible) de traitement : permet d'assurer "l'intégrité de base" d'un granule.

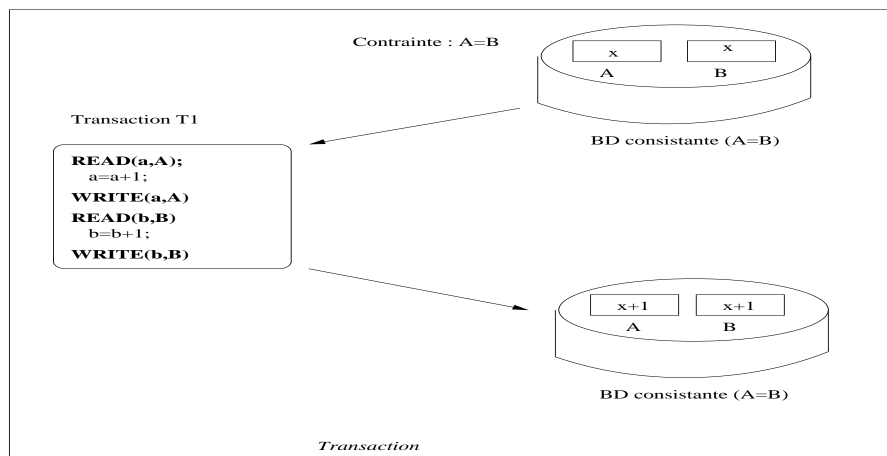


3.2.2 Transaction

Une transaction est une suite d'opération $\{a^1, a^2, ..., a^N\}$ qui préserve la consistence de la base de données.

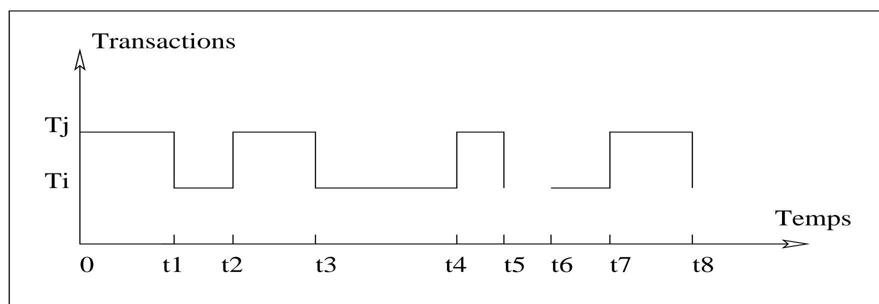


Exemple :



3.2.3 Transactions concurrentes

Deux transactions T^i et T^j sont dites concurrentes si leurs exécutions se recouvrent dans le temps comme illustré par la Figure ci-dessous :



Le diagramme suivant donne une représentation tabulaire

Temps	*Actif*	*En Attente ou Bloqué*
0	T^j	T^i
$t1$	T^i	T^j
$t2$	Tj	Ti
$t3$	Ti	Tj
$t4$	Tj	Ti
$t5$	idle	Ti Tj
$t6$	Ti	Tj
$t7$	Tj	$\oslash$
$t8$	idle	$\oslash$

Le processeur de requête exécute pendant une tranche de temps chaque transaction *prête*. Deux transactions concurrentes peuvent être soit :
- *indépendantes* : qui n'accèdent pas à des données communes
- *interractives* : qui accèdent à des données communes.

3.2.4 Etat d'une transaction

Une transaction est dite *active* (*running* à l'instant t quand elle dispose à cet instant t de toutes les ressources dont elle a besoin :

⇒ le processeur de requête effectue le traitement associé à son code à cet instant t.

Il y a au moins QUATRE raisons pour que cette activité soit interrompue :

1. fin-quantum : Elle a épuisé son *quantum* (tranche de temps durant laquelle l'ordonnaceur lui a alloué le processeur de requête : cet événement s'appelle *préemption*

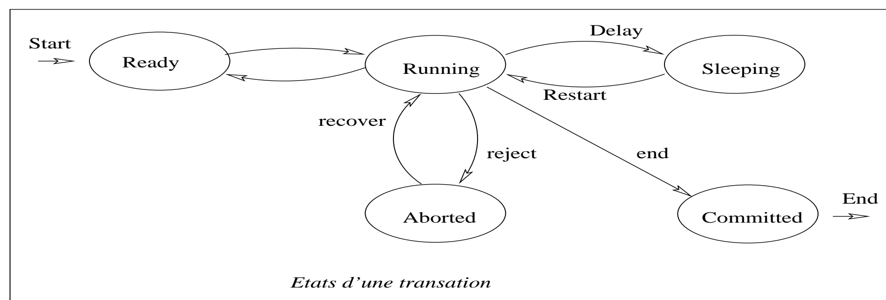
   ⇒ elle est dite *prête* à devenir active ou *ready*.
2. ressource non-disponible : Le processeur de requête a besoin d'une nouvelle ressource (données) non disponible. Elle doit alors attendre que la ressource soit de nouveau disponible :

   ⇒ elle est dite *en attente* de cette ressource ou *bloqué*, *sleeping*.
3. fin normale : L'exécution du code associé à cette requête est terminé normalement. Elle doit alors être confirmée :

   ⇒ elle est dite *committed*.
4. état incohérent : L'exécution du code associé à cette requête a engendré un état incohérent de la BD . Elle doit alors être défaite :

   ⇒ elle est dite *aborted*.

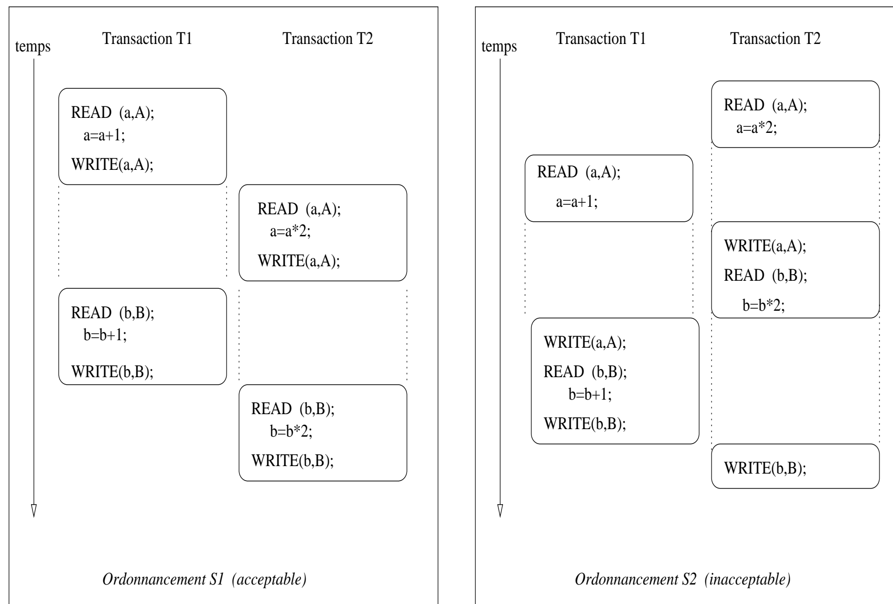
La Figure ci-dessous donne une représentation graphique de ces états élementaires d'une transaction et les différentes transitions possibles entre ces états.



Etats d'une transation

3.2.5 Ordonnancement des transactions

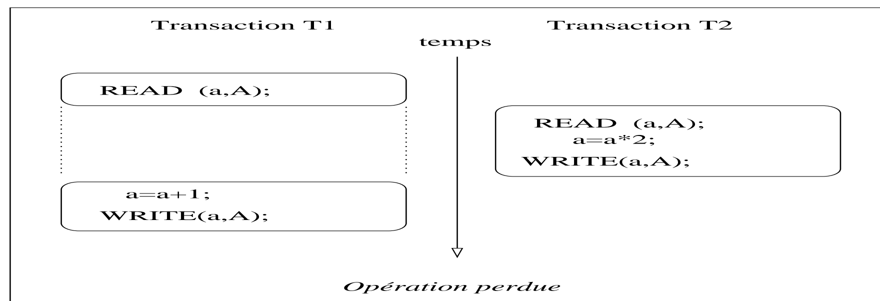
Un ordonnancement d'un ensemble de transactions $\{T^1, T^2, ..., T^N\}$ est une exécution concurrente des transactions $T^1, T^2, ..., T^N$



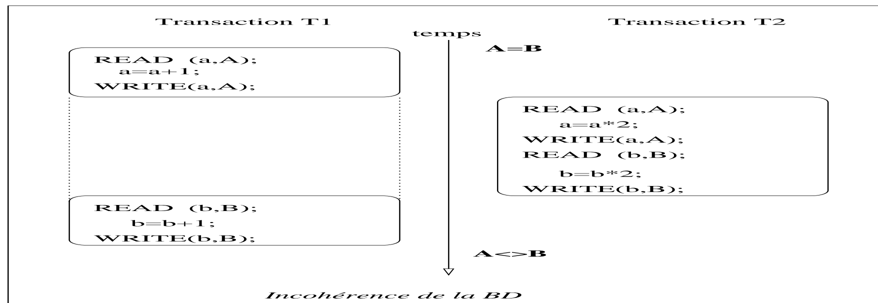
3.3 Caractéristiques des ordonnancements acceptables

3.3.1 Problèmes engendrés par les transactions concurrentes

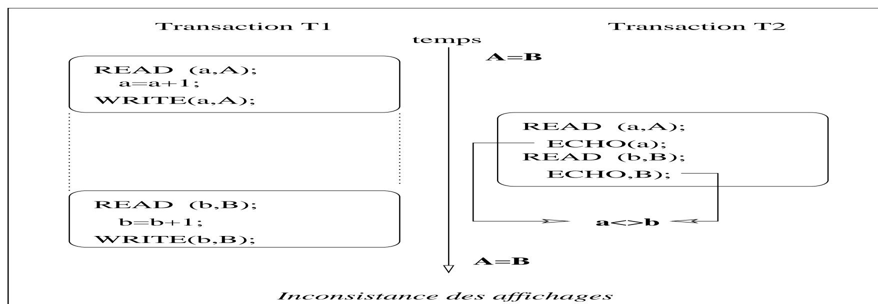
Opérations perdues



Incohérence de la base de données



Inconsistance des sorties (affichage)



3.3.2 Ordonnancement sérialisable

Un ordonnancement S d'un ensemble de transactions $\{T^1, T^2, ..., T^N\}$ est un ordonnancement série s'il existe une permutation π de $\{1, 2, ..., N\}$ telque :

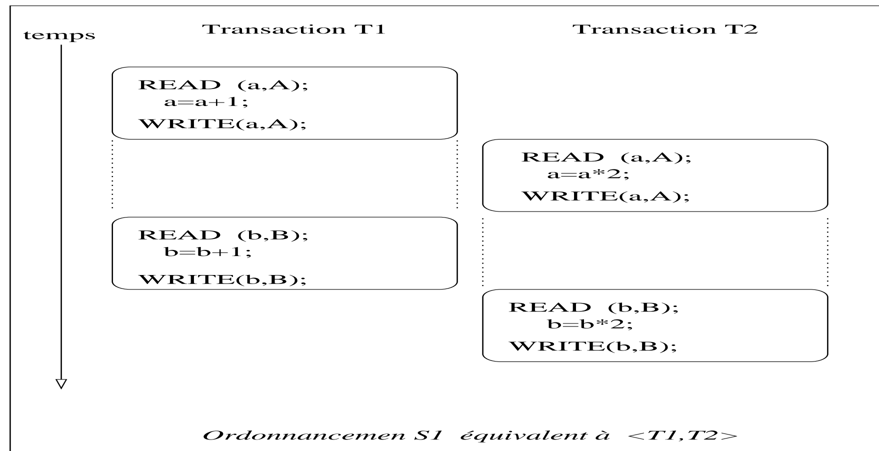
$S =< T^{\pi(1)}, T^{\pi(2)}, ..., T^{\pi(N)} >$.

Exemple : Pour 3 transactions $\{T^1, T^2, T^3\}$, on $3 != 6$ ordonnancements série possibles :

$$\begin{array}{l} S^1 =< T^1, T^2, T^3 > \\ S^2 =< T^1, T^3, T^2 > \\ S^3 =< T^3, T^1, T^2 > \\ S^4 =< T^3, T^2, T^1 > \\ S^5 =< T^2, T^1, T^3 > \\ S^6 =< T^2, T^3, T^1 > \end{array}$$

Un ordonnancement S d'un ensemble de transaction $\{T^1, T^2, ..., T^N\}$ est un ordonnancement sérialisable s'il est équivalent à un ordonnancement série de $\{T^1, T^2, ..., T^N\}$.

Exemple :



3.3.3 Caractéristique d'un ordonnancement sérialisable

Actions permutables

Deux actions A^i et A^j sont permutables si chaque exécution de $< A^i, A^j >$ est équivalent à l'exécution de $< A^j, A^i >$.

Exemple d'actions permutables :
- actions indépendantes : deux actions quelconques portant sur deux granules différents
- actions compatibles : deux lectures

Exemple d'actions non-permutables :
- une lecture et une écriture du même granule
- deux écritures du même granule
- deux insertion dans un B-index

La commutativité des actions effectués sur un granule est une propriété importante : évite le côntrole des accès concurrents à ce granule.

Relation de précédence

La transaction T^i précède la transaction T^j dans un ordonnancement $S =< T^1, T^2, ..., T^N >$ s'il existe deux actions non-permutables A^i et A^j telque A^i est exécutée par T^i avant A^j par T^j.

Graphe de précédence

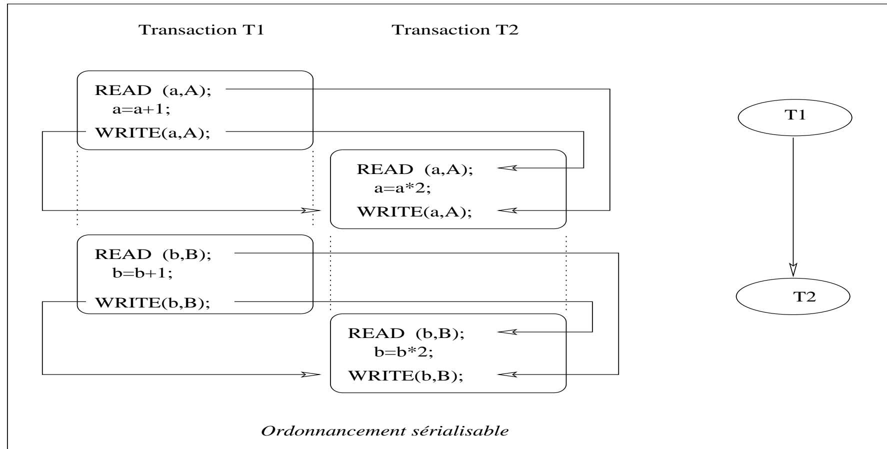
Un graphe de précédence d'un ordonnancement est un graphe orienté dont l'ensemble des noeuds est constitué par l'ensemble des transactions et telqu'il existe un arce de T^i à T^j si T^i précède T^j $(T^i < T^j)$.

Conditions suffisantes de sérialisation

1- Un ordonnancement est sérialisable s'il peut être transformé en un ordonnancement série par permuation des actions.

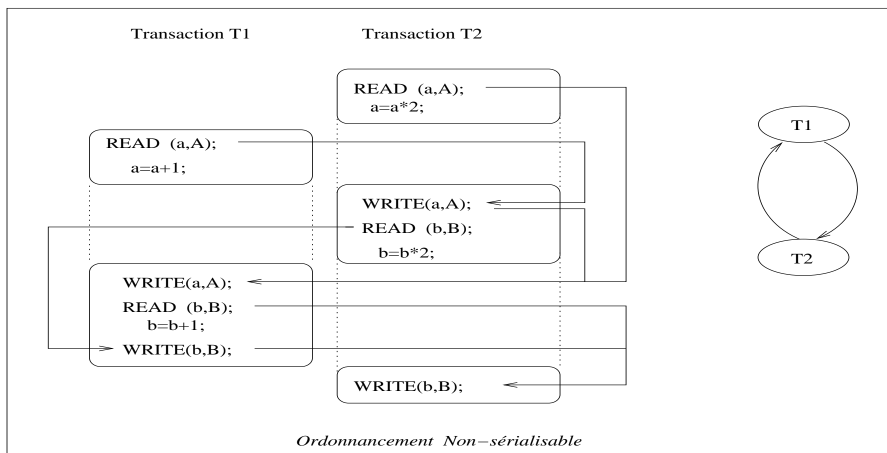
2- Un ordonnancement est sérialisable si son graphe de précédencxe n'a pas de circuit.

Exemple d'ordonnancement sérialisable :



Ordonnancement sérialisable

Exemple d'ordonnancement non-sérialisable :



Ordonnancement Non−sérialisable

Les précédences entre transactions sont introduites par des conflits $< READ^i, WRITE^j >$, $< WRITE^i, READ^j >$ et $< WRITE^i, WRITE^j >$ sur un même granule : Les algorithmes qui suivent, imposent un ordre sur les transactions qui doit être respecté par toutes les précédences introduites par des conflits ci-dessus. Cet ordre est basé sur :

- le moment du début de la transaction : initial ordering algorithm, cette méthode détecte les ordonnancements non-sérialisables (conflits) et relance une des transactions en conflit.
- l'arrivée du premier conflit : locking algorithm, cette méthode ne génère que des ordonnancements sérialisables en mettant dans l'état de sommeil (sleeping) une des transactions en conflit.

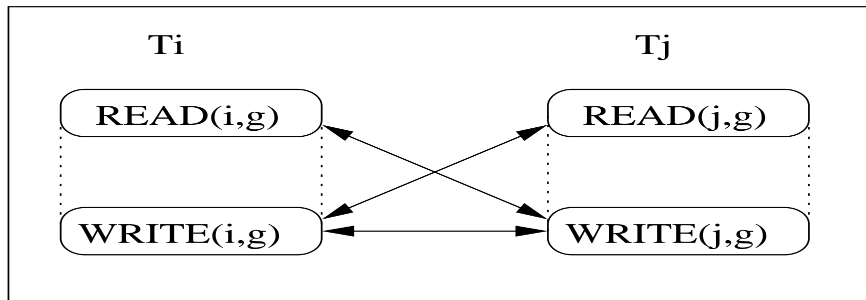


Figure 3.1 – Opérations non-permutables

3.4 Ordonnancement par estampillage à l'initialisation

3.4.1 Principe

1. Ordre total sur les transactions : ordonner les transactions en fonction du début de leur exécution,
2. Accès aux données : assurer que les accès aux données se font dans ce même ordre.
3. Lors d'un conflit d'accès une des transactions en conflit sera relancée.

3.4.2 Définition de l'ordre

L'ordre est un ordre arbitraire défini sous forme d'*estampille* à l'initialisation de la transaction.

- Estampille de Transaction :
  Une estampille de transaction est une valeur numérique unique assignée à une transaction.
- Implémenation :
  L'implémenation de l'estampilage peut se faire en utilisant :
  - un compteur système
  - l'horloge système

3.4.3 Estampille de granule

Pour garantir que les transactions opèrent sur les granules dans l'ordre défini par les estampilles : il est nécesaire de se rappeler l'estampille de la dernière transaction qui a opérer sur chaque granule.

Estampille de granule

Une estampille de granule est une variable numérique :

- assignée à un granule,
- qui mémorise l'estampille de la dernière transaction qui a opéré sur ce granule en :
  - lecture
  - écriture
  - etc

Table des estampilles

Les estampilles de granules sont gérées dans une table S. Une entrée de cette table connsiste en un couple :

1. un identificateur de granule,
2. valeur de l'estampille correspondant à ce granule.

En plus de cette table S, il est nécessaire de mémoriser dans une variable M, la valeur du plus grand estampille de granule.

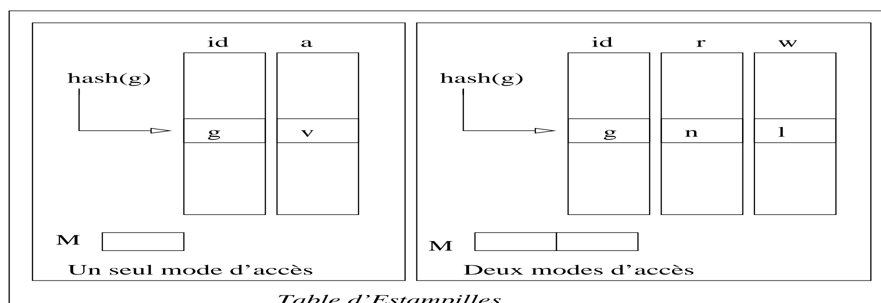


Table d'Estampilles

Détermination de l'estampille s d'un granule

1. Déterminer l'entrée i du granule g dans la table S,
2. Si g est dans S alors $s = S[i].a$ (estampille associé à g dans S),
3. Si g n'est pas dans S alors $s = M$ (estampille associé à la plus récente transaction).

3.4.4 Accès avec un ordre total

Cette méthode ne distingue pas entre les différents modes opératoires c'est à dire entre les opérations $READ$ et $WRITE$ permutables : un seul type d'estampille.

```
-------------------------------------------------------
 access(transaction i, granule g) {
    if (S[g] <= i) {
       S[g]=i;
       WRITE(g) ou READ(g);
    } else {
       abort(i);
    }
-------------------------------------------------------
```

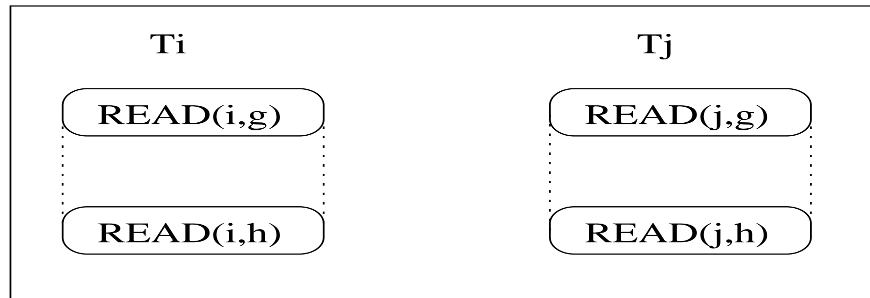
La table ci-dessous illustre les différents ordonnancements possibles de l'exemple de la $Figure$ (3.1). Avec la méthode d'accès ci-desussus seuls les ordonnancements (1) et (4) sont prévisibles

3.4.5 Accès avec un ordre partiel

On considère l'exemple ci-dessous qui ne présente aucun conflit (tous ses ordonnancements sont sérialisables) :

temps	Ordonnancements					
	1	2	3	4	5	6
t0						
t1	$READ^i$			$READ^j$		
t2	$WRITE^i$	$READ^j$		$WRITE^j$	$READ^i$	
t3	$READ^j$	$WRITE^i$	$WRITE^j$	$READ^i$	$WRITE^i$	$WRITE^j$
t4	$WRITE^j$	$WRITE^j$	$WRITE^i$	$WRITE^i$	$WRITE^j$	$WRITE^i$

FIGURE 3.2 – Ordonnancements possibles



et les différents ordonnancements possibles associés :

temps	Ordonnancements					
	1	2	3	4	5	6
t0						
t1	$READ^{i,g}$			$READ^{j,g}$		
t2	$READ^{i,h}$	$READ^{j,g}$		$READ^{j,h}$	$READ^{i,g}$	
t3	$READ^{j,g}$	$READ^{i,h}$	$READ^{j,h}$	$READ^{i,g}$	$READ^{i,h}$	$READ^{j,h}$
t4	$READ^{j,h}$	$READ^{j,h}$	$READ^{i,h}$	$READ^{i,h}$	$READ^{j,h}$	$READ^{i,h}$

La méthode précédente (voir point (3.4.4) ci-dessus) ordonne toutes les opérations sur les granules. De ce fait, seuls les ordonnancements (1) et (4) sont prévisibles alors que tous les ordonnancements sont acceptables.

Pour compléter cette lacune, il est seulement nécessaire d'ordonner les opérations non-permutables. Pour cela, on définit deux types d'estampilles :

1. w ou estampille write : estampille de la transaction qui a éffectué la plus récente écriture,
2. r ou estampille écriture : estampille de la transaction qui a éffectué la plus récente lecture.

On distingue alors deux primitives d'accès :

1. read : une lecture ne peut être en conflit qu'avec une écriture :

```
--------------------------------------------------------
 read(transaction i, granule g) {
    if (S[g].w <= i) {
       S[g].r = max(S[g].r, i);
       READ(g);
    } else {
       abort; // une des transactions en conflits
    }
--------------------------------------------------------
```

2. write : une écriture peut être en conflit aussi bien avec une lecture qu'une écriture :

```
--------------------------------------------------------
 write(transaction i, granule g) {
    if ( (S[g].w <= i) && (S[g].r <=i) ) {
       S[g].w =i;
       WRITE(g);
    } else {
       abort; // une des transactions en conflit
    }
--------------------------------------------------------
```

3.5 Ordonnancement par verrouillage à deux phases

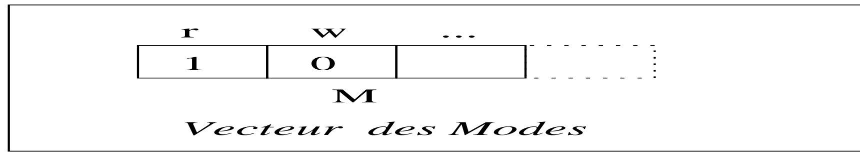
3.5.1 Principe

1. mode d'une opération : une opération est une suite d'actions effectuées par une transaction sur un même granule. A chaque opération est associée un mode opératoire.
2. Ordre partiel sur les opérations : permettre l'exécution des opérations compatibles c'est à dire constituées d'actions permutables.
3. Accès aux données : assurer que les accès aux données sont compatibles entre eux. Lors d'un conflit d'accès (incompatibilité) une des transactions en conflit sera bloquée.

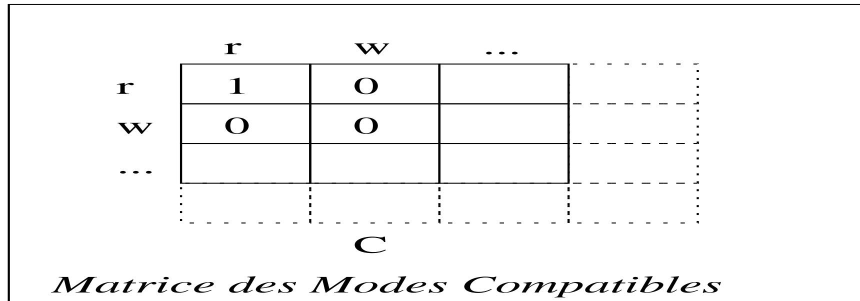
3.5.2 Matrice des modes compatibles

Un mode d'une opération est une propriété qui caractérise cette opération en déterminant sa compatibilité (permutabilité) avec les autres opérations, par exemple :
- mode r : en lecture (read),
- mode w : en écriture (write),
- mode i : insertion,
- mode u : mise à jour (update),
- etc ...

L'ensemble des modes est représenté par un vecteur M appelé vecteur des modes :



La compatibilité entre les différents modes est spécifiée par une matrice C :



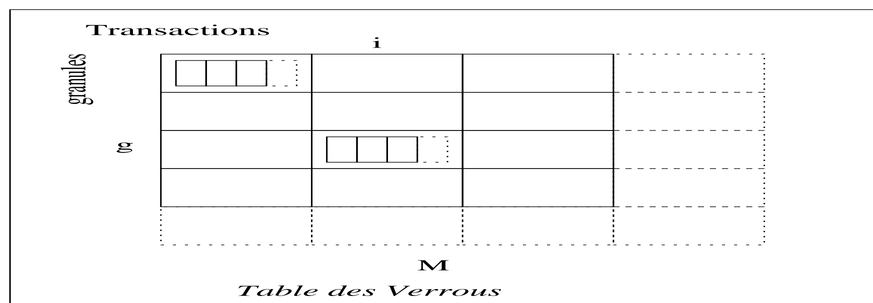
3.5.3 Protocole de verrouillage

Pour gérer les conflits, chaque opérateur doit poser un verrou sur une ressource avant de pouvoir l'utiliser. Ce protocole utilise :

1. Table des verrous
2. Files d'attente de granule
3. Primitives LOCK et UNLOCK

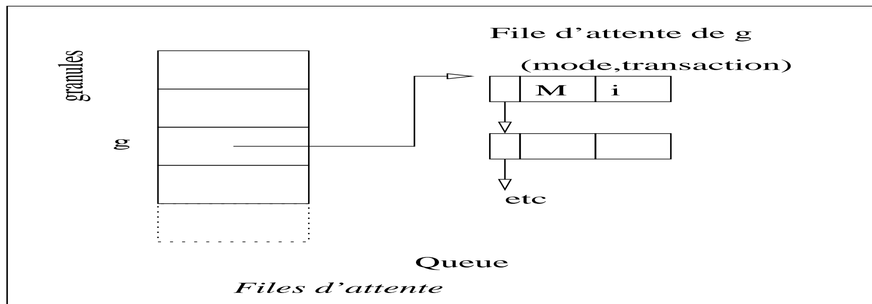
Table des verrous

La table des verrous $A[g, i]$ mémorise pour un granule g, les verrous posés par une transaction i sur g. Une entrée de la table A est une instance du vecteur des modes M comme illustré par la figure ci-dessous :



Files d'attente de granule

Lors d'un conflit d'accès (incompatibilité), une des transactions en conflit sera bloquée c'est à dire qu'elle passe de l'état *active* à l'état *bloquée*. Les transactions bloquées sont réparties dans des files d'attentes *Queue*, en fonction du granule sollicité par la transaction lors du conflit :



Files d'attente

Primitives LOCK et UNLOCK

Chaque transaction t effectuant une opération en mode m sur un granule g doit être parenthésé par les deux primitives :

- $LOCK(t, g, m)$ permet à une transaction t de :
  1. poser un verrou m sur un granule g si m est comptaible, ou
  2. se mettre en sommeil (c'est à dire d'attendre que le verrou m sur g devient possible) si m n'est pas compatible.
- $UNLOCK(t, g)$ permet à une transaction t de lever le verrou qu'elle a auparavant posé sur g.

comme suit :

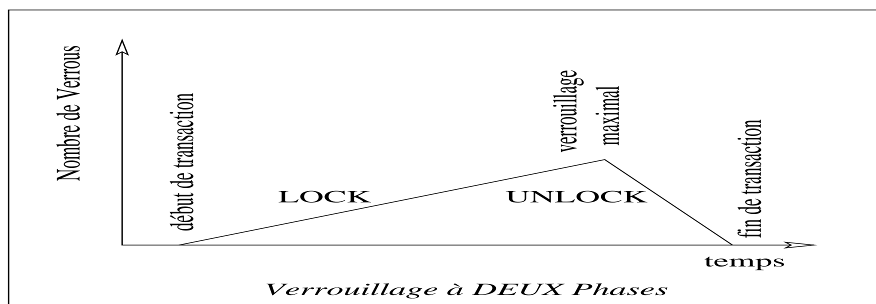
```
------------------------------
 LOCK(t,g,m)
    operation(g)
 UNLOCK(t,g)
------------------------------
```

Une transaction bien parenthésés est appelée transaction *bien formée*.

Condition de sérialisation

Pour que les ordonnancement prévisibles par ce protocole soient sérialisables, il faut que les transactions bien formées de ces ordonnacements opèrent en *deux phases*.

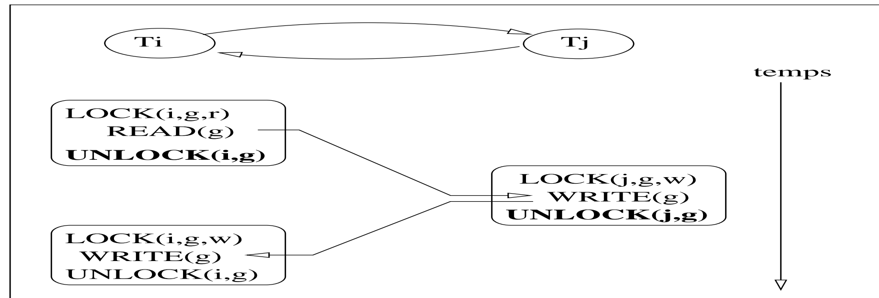
Une transaction à *deux phases* est une transaction bien formée dont chaque primitive $LOCK$ précède l'ensemble des primitives $UNLOCK$ comme illustré par la figure :



Verrouillage à DEUX Phases

Preuve :

supposons un ordonnacement de transactions à deux phases $< T^i, T^j >$ non sérialisable c'est à dire contenant un *cycle* comme illustré par le graphe de précédence ci-dessous :



alors, si le graphe de précédence comporte un cycle cela signifie que T^i et T^j n'opèrent pas en deux phases.

3.5.4 Problème de Blocage mutuel

Exemple

Si on reprend les requête de la $Figure(3.1)$ et les ordonnancements correspondants de la $Figure(3.2)$, on constate que :

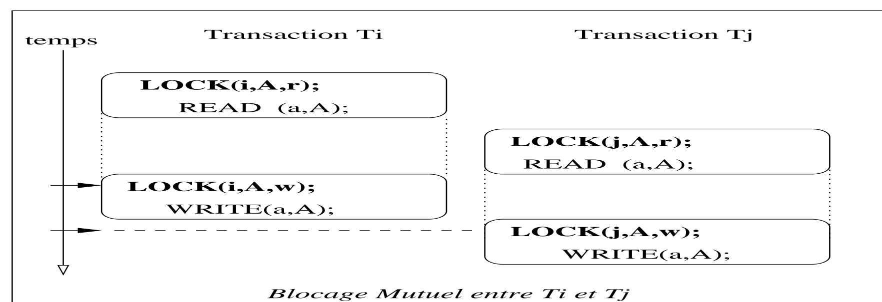
1. les ordonnancements prévisibles sont : les ordonnancements sérialisables (1) et (4),
2. Par contre les ordonnancements non-sérialisables : (2) , (3), (5) et (6) crée une situation de blocage mutuel juste après l'étape (t2) c'est à dire que le protocole d'ordonnancement est incapable de dénouer la situation : les deux transactions sont dans un état *attente* infinie.

Définition

Un blocage mutuel est un état du système dans lequel :
- un ensemble de transactions sont bloquées : chacune attend un granule,
- aucune transaction du système : ne peut débloquer ces transactions.

Illustration du blocage mutuel

Sur l'ordonnancement (2) de l'exemple de la $Figure(3.2)$



Blocage Mutuel entre Ti et Tj

Graphe d'attente des granules

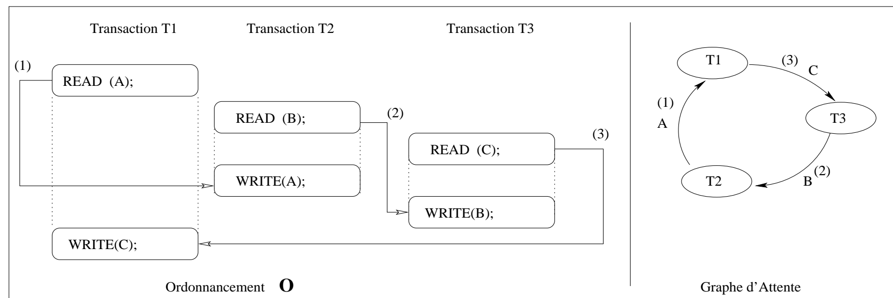
Un graphe d'attente est un graphe orienté $G(T, W)$ où
- l'ensemble des sommets : est l'ensemble T des transactions,
- l'ensemble des arcs : Il y a un arc de T^i à T^j étiqueté g si T^i attend T^j pour accéder au granule g c'est à dire que les opérations de T^i et T^j sur g ne sont pas compatibles.

Propositions :

Il y a un blocage mutuel si le graphe d'attente contient un cycle.

Si on change l'orientation de chaque arc du graphe d'attente , on obtient un sous-graphe du graphe de précédence.

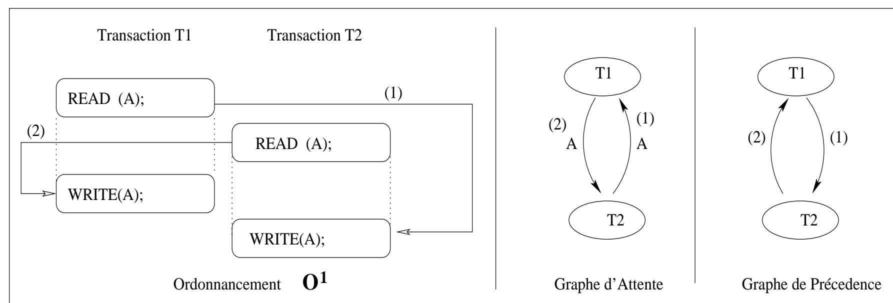
Exemple



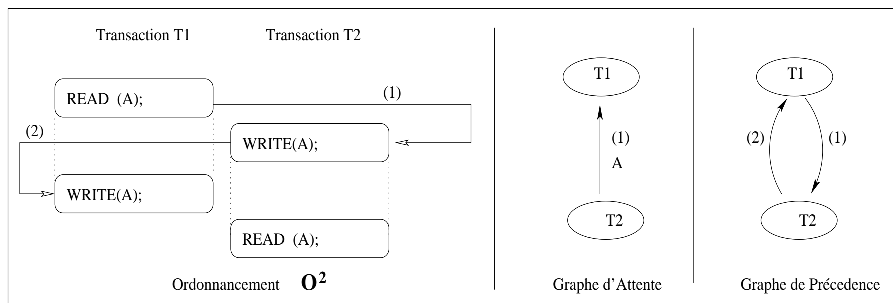
A

Graphe d'attente et Sérialisation

Exemple 1



Exemple 2



3.5.5 prévention du blocage mutuel

Une approche de la prévention du blocage mutuel consiste à pré-ordonner les transactions de sorte à éliminer la situation qui amène au blocage mutuel lors de l'ordonnancement par verrouillage à deux phases.

Cette ordre sur les transactions :
- consiste à mettre des *estampilles (timestamps)* sur les transactions,
- permet d'éliminer les cycles dans les graphes de précédence

Deux méthodes permettent de mettre en oeuvre cette approche :

1. DIE-WAIT
2. WOUND-WAIT

DIE-WAIT

Si une transaction t veut accéder à un granule g détenu par une transaction j plus jeune que t (timestamp de t plus petit que celui de j) alors t se bloque sinon t (plus jeune) est avortée. Il est nécessaire d'avoir une table d'allocation des granules. Si la transation t est avortée, elle est relancée avec la même estamplille pour garder bénéfice de son age (permet d'éviter la famine).

```
-------------------------------------------------------
 DIE-WAIT(t,g) {
    j = la transaction détenant g
    if (  t < j ){
          insérer [t, M] dans Queue[g] ;
          bloquer la transaction t ;
    } else {
          abort t ;
    }
 )
-------------------------------------------------------
```

WOUND-WAIT

WOUND-WAIT est l'approche symétrique de DIE-WAIT. Si une transaction t veut accéder à un granule g détenu par une transaction j plus vieille que t (timestamp de t plus grand que celui de j) alors t se bloque sinon t (plus agée) remplace j (même timestamp) , j est avortée.

```
-------------------------------------------------------
 WOUND-WAIT(t,g) {
    j = la transaction détenant g
    if (  t > j ){
          insérer [t, M] dans Queue[g] ;
          bloquer la transaction t ;
    } else {
          abort j ;
          remplacer j par t ;
    }
 )
-------------------------------------------------------
```

3.5.6 Détection du blocage mutuel

- Vérifier que le graphe d'attente ne contient pas de cycle.
- La vérification peut se faire dès que le durée du sommeil d'une transaction (état *sleeping*) dépasse un seuil fixé.