

Bases de données

Cours 6 : Introduction à des notions avancées (Index, Déclencheurs, Transactions)

Nicolas DURAND

Polytech'Marseille
Université d'Aix-Marseille
nicolas.durand@univ-amu.fr
<http://nicolas.durand.perso.luminy.univmed.fr/pub/>

2012-2013

Préambule

Exemple utilisé dans ce cours

Client(numClient, nom, prénom, ca)

Article(numArticle, description, prixUnitaire, quantitéEnStock)

Commande(numCommande, #numClient, dateCommande, montant)

LigneCommande(#numCommande, #numArticle, quantité)

Sommaire

- 1 **Index**
 - Définition, caractéristiques et 1er exemple
 - Création et suppression d'un index
 - Mécanismes
- 2 Déclencheurs (Triggers)
- 3 Transactions, reprise, concurrence d'accès

Définition

- Pour compléter le schéma d'une table (relation), on peut créer des **index**.
- Permet une **recherche rapide d'une information**.
- Notion d'index : même que pour un livre

Livre	Base de données
Mots-clés de l'index	Valeurs d'un attribut
Pages	Adresses mémoires où sont stockées ces valeurs

Caractéristiques

- Objet optionnel associé à une table.
- Mis à jour automatiquement lors de modifications de la table.
- Possibilité de créer plusieurs index sur une même table.
- Peut concerner plusieurs attributs d'une même table (*index composé*).

Exemple

Exemple

```
SELECT * FROM Client WHERE nom = 'Martin' ;
```

- Un moyen pour récupérer la ou les lignes répondant à la clause nom='Martin' est de balayer toute la table.
- Le temps de réponse sera prohibitif dès que la table dépasse quelques centaines de lignes.
- Afin d'optimiser ce type de requête, on pourra indexer l'attribut "nom".

Création et suppression d'un index

Création

```
CREATE INDEX nom_index ON nom_table (nom_attribut1, ...);
```

Exemple

```
CREATE INDEX idxNomClient ON Client (nom);
```

Suppression

```
DROP INDEX nom_index;
```

- Création **implicite** d'index : chaque fois qu'une **clé primaire** ou une **contrainte d'unicité** sur un attribut est définie pour une table.
- Pour vérifier l'existence d'index : consulter les vues (ORACLE) USER_INDEXES, USER_IND_COLUMNS, ALL_INDEXES, ALL_IND_COLUMNS ou DBA_INDEXES.

Critères d'indexation

On choisira de créer un index sur :

- Les attributs utilisés comme **critère de jointure**,
- Les attributs servant souvent de **critères de sélection**,
- Sur une table de gros volume (d'autant plus intéressant si les requêtes sélectionnent peu de lignes).

L'adjonction d'index accélère la recherche des lignes.

Mais il ne faut pas créer des index à tort et à travers.

- Impact négatif sur les commandes d'insertion et de suppression (car mise à jour de tous les index portant sur la table) → coût.

Critères d'indexation

Remarque :

- Les *valeurs non renseignées* ne sont pas stockées dans l'index (pour minimiser le volume), l'index n'est donc d'aucune utilité pour retrouver les valeurs non renseignées (lorsque le critère de recherche est IS NULL ou IS NOT NULL).
- L'index n'est pas mis en oeuvre lors de l'évaluation d'une requête si la(les) attribut(s) correspondant(s) sont utilisés par l'intermédiaire d'une *expression*, d'une *fonction* ou d'une *conversion implicite*.

Exemples

- La table "Client" bénéficie d'un index sur l'attribut "nom", d'un index sur "numClient" et d'un index sur le champs "ca".
- Les index sont utilisés dans les cas suivants :

```
SELECT * FROM Client WHERE numClient = 500 ;  
SELECT * FROM Client WHERE numClient >= 412 ;  
SELECT * FROM Client WHERE numClient BETWEEN 300 AND 500 ;  
SELECT * FROM Client WHERE nom = 'Martin' ;  
SELECT * FROM Client WHERE nom LIKE 'M%' ;
```

- Les index sont inutiles dans les cas suivants :

```
SELECT * FROM Client WHERE nom IS NULL ;  
SELECT * FROM Client WHERE ca*10 >= 10000 ;
```

Mécanisme d'accès aux enregistrements d'un index

- Mécanisme interne au SGBD.
- Organisation arborescente
 - Séquentiel-indexé,
 - Arbres B+ (par défaut dans ORACLE)
- Accès par hachage
 - Statique,
 - Dynamique

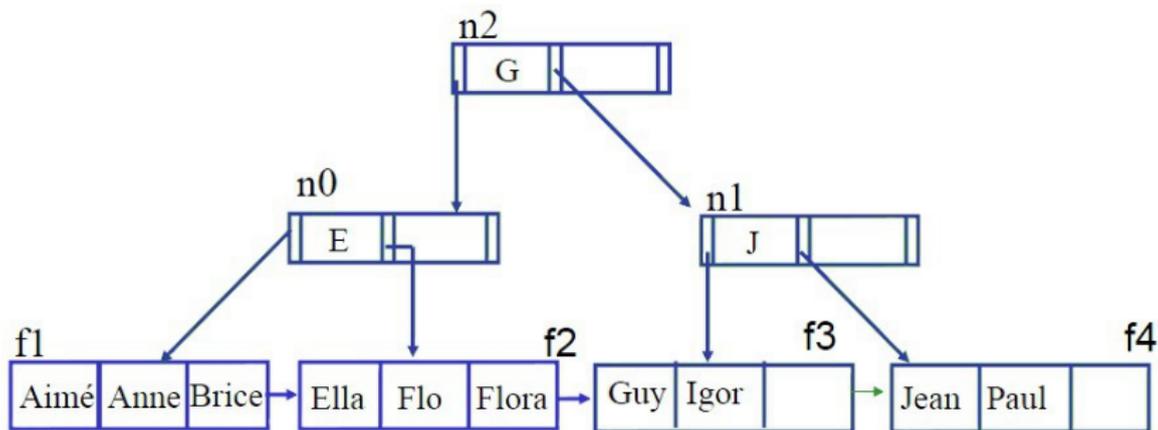
Arbres B+

- Les arbres B ont été introduits par Bayer et McCreight en 1972 et ont fait l'objet de nombreux développements par la suite
- Nous décrivons une variante des arbres B : les arbres B+, qui sont très largement utilisés pour construire des index de BD
- Un arbre B+ d'ordre m (entier impair ≥ 3) est un arbre équilibré dont chaque noeud est enregistré dans une page stockée sur disque

Arbres B+ : organisation

- Une feuille contient une séquence d'enregistrements triée par ordre croissant de clé
- Une feuille est au moins à moitié remplie sauf si elle est l'unique noeud de l'index
- Les feuilles sont chaînées entre elles dans l'ordre de leur première clé à l'aide du pointeur p .
- Un noeud non terminal contient une séquence d'enregistrements triée par ordre croissant de clé
- Un noeud est au moins à moitié rempli sauf s'il est la racine

Arbres B+ : exemple



Recherche de la clé "Igor" :

- $I > G$, alors **n1**
- $I < J$, alors **f3**,
- $Igor \in f3$

Hachage

- Le hachage repose sur la construction d'une fonction dite de hachage qui appliquée à la clé d'un enregistrement fournit l'adresse de cet enregistrement
- Le hachage est dit **statique** ou **dynamique** selon la fonction de hachage est fixée ou évolue durant la vie de l'index
- Un index à accès par hachage peut être organisé avec ou sans répertoire

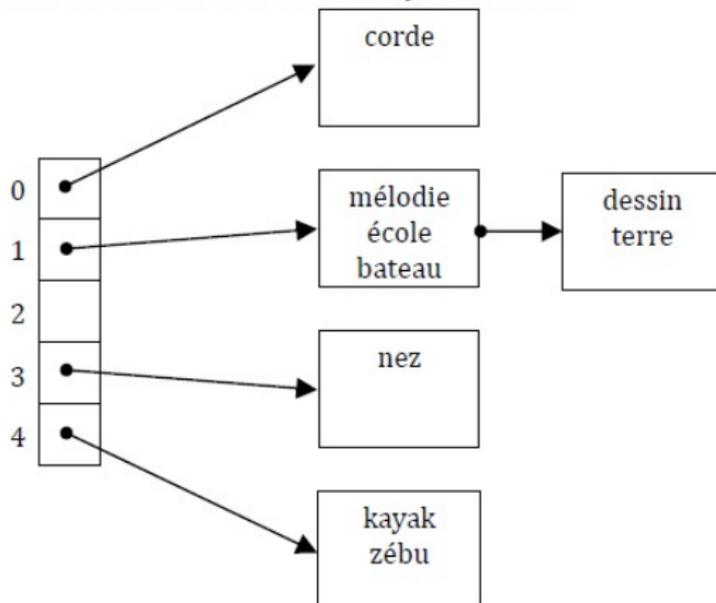
Hachage statique avec répertoire

$N = 5$: répertoire de 5 cases,

3 enregistrements au maximum,

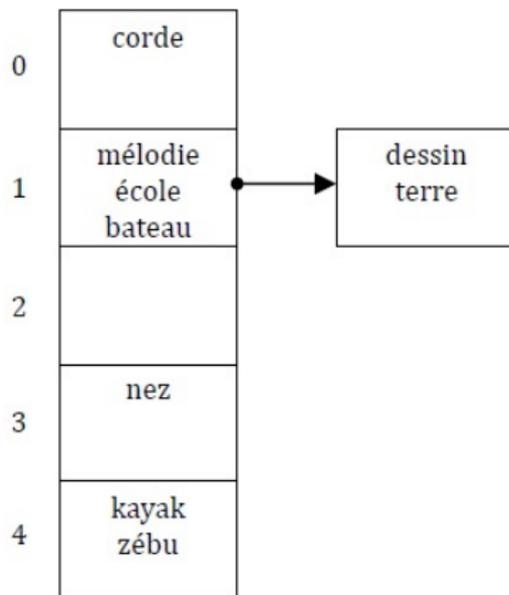
$h(c) = (\text{somme des codes ASCII des caractères de } c) \text{ modulo } 5$.

c	$h(c)$
mélodie	1
école	1
nez	3
bateau	1
kayak	4
zébu	4
dessin	1
corde	0
terre	1



Hachage statique sans répertoire

On peut éviter l'utilisation d'un répertoire en créant un index de N pages contiguës. Le code haché donne alors directement accès à la page contenant les clés recherchées.



Sommaire

- 1 Index
- 2 Déclencheurs (Triggers)
 - Introduction
 - Caractéristiques
 - Création et suppression
 - Exemples
- 3 Transactions, reprise, concurrence d'accès

Définition

- Un **déclencheur** (ou triggers) est une règle, dite *active*, de la forme : "événement-condition-action".
- **Procédure stockée** dans la base qui est déclenchée automatiquement par des *événements spécifiés par le programmeur* et ne s'exécutant que lorsqu'une condition est satisfaite.

Utilité

Les déclencheurs permettent :

- La possibilité d'**éviter les risques d'incohérence** dus à la présence de redondance.
- L'**enregistrement automatique** de certains événements.
- La spécification de **contraintes liées à l'évolution de données**.
 - Exemple : un salaire ne peut qu'augmenter.
- De définir toutes **règles complexes liées à l'environnement d'exécution** (restrictions sur des horaires, des clients, ...).

Principe

Séquence *Événement-Condition-Action* :

- Trigger déclenché par un **événement** spécifié par le programmeur
 - Insertion, destruction, modification sur une table.
- Test de la **condition** : si cette dernière n'est pas vérifiée, alors l'exécution s'arrête.
- Si vérifiée, l'**action** est réalisée (toutes opérations sur la base de données).

Caractéristiques

- **Un seul déclencheur par événement sur une table.**
- Les déclencheurs permettent de rendre une base de données **dynamique**.
 - Une opération peut en déclencher d'autres, qui elles-mêmes peuvent entraîner en cascade d'autres déclencheurs. . .
- ***Ce mécanisme n'est pas sans danger !***
 - Risque de boucle infinie.

Caractéristiques

- *Manipulation simultanée* de l'**ancienne** et de la **nouvelle** valeur d'un attribut (→ tests sur l'évolution).
- Un déclencheur peut être exécuté :
 - **Une fois** pour un seul ordre SQL,
 - Ou **à chaque ligne concernée** par cet ordre.
- L'action peut être réalisée **avant** ou **après** l'événement.

Caractéristiques

Concernant "**action**" :

- SQL n'est pas procédural
 - Les SGBD fournissent une extension du langage SQL
 - instructions de branchement conditionnel, instructions de répétition, affectations, ...
- Langage impératif permettant de créer des véritables procédures (procédures stockées)
 - **PL/SQL** pour ORACLE
 - PL/pgSQL pour PostgreSQL
 - T-SQL pour SQL Server
 - norme SQL2003 pour DB2 et MySQL5.

Création et suppression (ORACLE)

```
CREATE [OR REPLACE] TRIGGER nom_trigger  
BEFORE | AFTER  
INSERT | DELETE | UPDATE [OF col,..,col]  
[OR INSERT | DELETE | UPDATE [OF col,..,col] ...]  
ON nom_table  
[[REFERENCING [OLD [AS] old] [NEW [AS] new]]  
[FOR EACH ROW] [WHEN (condition)]]  
bloc_pl/sql ;
```

Recompilation : ALTER TRIGGER nom_trigger COMPILE ;

Activation/désact. : ALTER TRIGGER nom_trigger {ENABLE |
DISABLE} ;

Suppression : DROP TRIGGER nom_trigger ;

Création (ORACLE)

- **REPLACE** : lorsque l'on veut installer une nouvelle version du trigger
- **BEFORE/AFTER** : précise le moment de déclenchement du trigger (avant ou après)
- **DELETE/UPDATE/INSERT** : précise le ou les évènements concernés par le déclenchement. S'il y a plusieurs évènements, on les sépare avec "OR".
- **ON TABLE** précise le nom de la table concernée.
- **FOR EACH ROW** : précise si le trigger doit être déclenché pour chaque ligne mise à jour. Dans ce cas, il existe deux variables *:new* et *:old* qui contiennent respectivement le nouveau et l'ancien contenu de la ligne.
- **WHEN(condition)** : l'action peut n'être déclenchée que pour une partie des lignes mises à jour.

Déclencheur : exemple 1

Vérification qu'un prix ne peut pas baisser.

```
CREATE OR REPLACE TRIGGER pas_baisse_prix
BEFORE UPDATE OF prixUnitaire ON Article
FOR EACH ROW
WHEN (OLD.prixUnitaire > NEW.prixUnitaire)
BEGIN
    raise_application_error(-20100, "le prix d'un article ne peut pas
diminuer");
END;
```

Déclencheur : exemple 2

Déclencheur insérant un enregistrement à l'intérieur d'une seconde table "Client100" lorsqu'une opération d'insertion s'est accomplie dans une première table "Client".

Il vérifie aussi si le nouveau n-uplet possède un attribut "ca" supérieur ou égal à 100.

```
CREATE TRIGGER declencheur1
AFTER INSERT ON CLIENT
FOR EACH ROW
WHEN (NEW.ca >= 100)
BEGIN
    INSERT INTO Client100
        VALUES( :new.numClient, :new.nom, :new.prénom, :new.ca) ;
END ;
```

PL/SQL

- Procedural SQL
- Bloc PL/SQL

```
DECLARE
-- déclarations de variables locales
BEGIN
/* instructions, commandes SQL, structures de contrôle */
EXCEPTION
/* gestion des erreurs */
END;
```

DECLARE est facultatif.

BEGIN...END est obligatoire.

EXCEPTION est facultatif.

Déclencheur : exemple 3

Si insertion nouvelle commande alors mise à jour de Client pour assurer la cohérence de la base.

```
CREATE OR REPLACE TRIGGER maj_client
BEFORE INSERT ON Commande
FOR EACH ROW
DECLARE
  Vclient Client.numClient%TYPE ;
BEGIN
  SELECT numClient INTO Vclient FROM Client
  WHERE numClient = :new.numClient ;
  UPDATE Client SET ca = ca + :new.montant
  WHERE numClient = :new.numClient ;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
  INSERT INTO Client
  VALUES ( :new.numClient, NULL, NULL, :new.montant ) ;
```

Sommaire

- 1 Index
- 2 Déclencheurs (Triggers)
- 3 **Transactions, reprise, concurrence d'accès**
 - Transactions
 - Reprise
 - Concurrence d'accès

Transaction

- Transaction = *séquence d'opérations* qui accèdent et modifient le contenu d'une base de données.
- *Unité de traitement atomique* qui fait passer la base de données d'un état **cohérent** à un état **cohérent**. Pendant la transaction, l'état de la base de données peut être incohérent.

Une transaction s'effectue :

- **Complètement**, les mises à jour qu'elle a effectuées sur la base de données sont **validées**.
- **Incomplètement** (annulation ou panne), toutes les mises à jour effectuées depuis le début de la transaction sont **invalidées**.

Etapes d'une transaction

- Délimitation par des instructions de début et de fin.
- Début si :
 - Connexion à la base de données, ou fin de la transaction précédente.
- Fin si :
 - *COMMIT* (validation de la transaction), *ROLLBACK* (abandon de la transaction), ou fin (normale) de session.

START

...suite d'opérations...

COMMIT

START

...suite d'opérations...

ROLLBACK

- Découpage d'une transaction en créant des points d'arrêt :
SAVEPOINT point ;
- Annulation d'une partie de la transaction :
ROLLBACK TO [SAVEPOINT] point ;

Transaction : exemple

```
BEGIN
  INSERT INTO Client2 (nomClient) VALUES ('acheteur');
  COMMIT ;
  afficher('Le client a été inséré');
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    afficher('Le client existe déjà');
    ROLLBACK ;
  WHEN OTHERS
    afficher('Erreur inconnue à l'insertion');
    ROLLBACK ;
END ;
```

Propriétés (ACID) d'une transaction

Le SGBD doit assurer que toute transaction possède les propriétés suivantes :

- **Atomicité**

- Une transaction est une unité atomique de traitement.

- **Cohérence**

- Une transaction préserve la cohérence de la BD.

- **Isolation**

- Les exécutions des transactions ne doivent pas interférer les unes avec les autres.

- **Durabilité**

- Les changements appliqués à la BD par une transaction validée doivent persister (même suite à une défaillance).

Récupération de bases de données (Reprise)

- **Pannes** d'un SGBD :
 - panne d'ordinateur,
 - panne de disque.
- La **reprise à chaud**, après un abandon de transaction ou une panne d'ordinateur, peut être réalisée automatiquement et rapidement, en s'appuyant sur la tenue d'un journal qui garde en mémoire tous les événements d'une transaction.
- La **reprise à froid**, après une panne de disque est plus longue à mettre en oeuvre. Elle nécessite de réaliser des copies régulières de la BD et un archivage des mises à jour entre deux copies.

Atomicité et durabilité

- Le respect de l'atomicité peut impliquer de défaire les effets d'une transaction lorsque celle-ci a été abandonnée.
- Le respect de la durabilité implique que le SGBD doit être capable de remettre la base de données en état après une panne :
 - les mises à jour faites par une transaction non confirmée avant la panne doivent être défaites.
- C'est le **gestionnaire de reprise** qui assure cette tâche.

Journalisation

- **Journal** : fichier séquentiel qui contient une suite d'enregistrements dont chacun décrit un événement concernant la vie des transactions et les modifications de la BD.
- Fichier séquentiel enregistré sur une mémoire stable, c.-à-d. une mémoire qui théoriquement ne peut pas être détruite.
- Si nécessaire, le journal est sauvegardé régulièrement sur un disque miroir, bandes magnétiques, . . .

Exemple d'événements journalisés

Événement	Signification
$(T \text{ start})$	début de la transaction d'identificateur T
$(T D a)$	mise à jour la donnée D par la transaction T : son ancienne valeur était a
$(T D n)$	mise à jour la donnée D par la transaction T : sa nouvelle valeur est n
$(T \text{ commit})$	confirmation de la transaction T
$(T \text{ abort})$	abandon de la transaction T
(check point)	point de reprise

Un extrait de journal

T_1	T_2
start	
read A	
A = A + 10	
write A	
rollback	
	start
	read B
	B = B + 20
	write B
	commit

Journal
T_1 start
T_1 A 30 40
T_2 start
T_2 B 70 90
T_2 commit

Point de reprise (check point)

- **Point de reprise** = marque dans le journal indiquant un moment où :
 - Aucune transaction n'était en cours.
 - Toutes les données écrites par des transactions antérieures au point de reprise avaient été transférées sur disque.
- Pour obtenir un point de reprise, il faut donc :
 - Interdire le début de nouvelles transactions,
 - Laisser se terminer les transactions en cours.

Concurrence d'accès

- SGBD : **multi-utilisateurs**.
- Une même information peut manipulée par plusieurs utilisateurs à la fois.
→ ***problèmes d'accès concurrents***
- Unité de la concurrence = **transaction**.

Cas d'un système de réservation

- Alice et Tom travaillent chacun dans une agence de voyage.
- Alice (transaction T1) veut annuler N réservations sur un vol V1 et réserver N places sur un vol V2.
- Tom (transaction T2) veut réserver M places sur le vol V1.
- L'exécution concurrente de T1 et de T2 *sans contraintes de synchronisation* peut produire un certain nombre de problèmes.

Problèmes dus à la concurrence

Si aucun contrôle du déroulement, les problèmes suivants peuvent apparaître :

- Perte de mise à jour.
- Lecture impropre.
 - Lecture de données incohérentes,
 - Lecture de données non confirmées.
- Lecture non reproductible.
- Objets fantômes.

Perte de mise à jour

T_1	T_2	BD
		$A = 10$
read A		
	read A	
$A = A + 10$		
write A		$A = 20$
	$A = A + 50$	
	write A	$A = 60$

On devrait avoir $A=70$ MAIS on a $A = 60$: perte de la mise à jour de T_1 .

Lecture impropre (données incohérentes)

T_1	T_2	BD
		$A + B = 200$
		$A = 120$ $B = 80$
read A		
$A = A - 50$		
write A		$A = 70$
	read A	
	read B	
	display A + B (150 est affiché)	
read B		
$B = B + 50$		
write B		$B = 130$

T_1 est cohérente MAIS T_2 devrait afficher la valeur 200.

Lecture impropre (données non confirmées)

T_1	T_2	BD
		A = 50
	A = 70	
	write A	A = 70
read A (70 est lu)		
	rollback (La valeur initiale de A est restaurée)	A = 50

T_1 a lu une valeur de A incorrecte : tout doit se passer comme si T_2 n'avait jamais changé A.

Lecture non reproductible

T_1	T_2	BD
		A = 10
	read A (10 est lu)	
A = 20		
write A		A = 20
	read A (20 est lu)	

T_2 (qui ne modifie pas A) devrait obtenir à chaque lecture la même valeur pour cette donnée.

Objet fantôme

T_1	T_2	BD
		$E = \{1, 2, 3\}$
display card(E) 3 est affiché		
	insert 4 into E	$E = \{1, 2, 3, 4\}$
display card(E) 4 est affiché		

T_1 n'a pas vu l'ajout de 4 dans E par T_2 , pour T_1 4 est un objet fantôme.

Résolution des problèmes de concurrence

- Mécanismes internes au SGBD pour gérer ces problèmes.
- Etude de la "sérialisabilité" de l'exécution d'un ensemble de transactions,
- Verrouillage : n-uplet (toutes ses valeurs), table (toutes ses lignes), BD,
- Verrou partagé, verrou exclusif.

Niveau d'isolation d'une transaction

Niveaux d'isolation :

- serializable : aucun problème. (attention. . .)
- read committed (par défaut pour ORACLE) : pas de perte de mise à jour, ni de lecture impropre.
- read uncommitted (absent d'ORACLE) : pas de perte de mise à jour.
- read only

```
SET TRANSACTION <option> ;
```

```
<option> ::= read only | isolation level <niveau-d-isolation>
```

```
| read write | use rollback segment <rollback_segment>
```

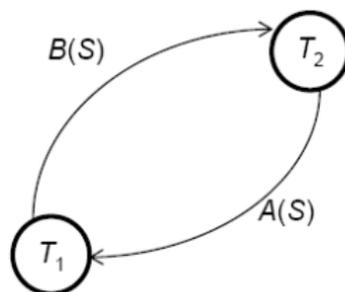
```
<niveau-d-isolation> ::= serializable | read committed
```

```
SET TRANSACTION NAME ma_trans ;
```

```
SET TRANSACTION ISOLATION LEVEL ma_trans SERIALIZABLE ;
```

Interblocage

T_1	T_2
lock X A	
	lock X B
lock S B	
attente	lock S A
attente	attente



graphe d'attente

Résolution de l'interblocage :

- Prévention, détection.
- Estampillage : transactions estampillées avec l'heure de lancement (TimeStamp, TS). *die-wound*, *wound-wait*.
- Autres stratégies : *no waiting*, *cautious waiting*, *timeout*.