# Cours de bases de données, aspects systèmes, http://sys.bdpedia.fr

## Verrouillage ‘a deux phases

Philippe Rigaux

le cnam

# Verrouillage à deux phases

Le principal (et le plus ancien) algorithme pour assurer la sérialisabilité.

Est réputé engendrer beaucoup de blocages et de rejets.

Repose sur le verrouillage (surprise !) des nuplets.

# Les verrous

Pose de **verrous** sur les nuplets.

- Le verrou **partagé** (*shared lock*) autorise la pose d'autres verrous partagés sur le même tuple.
- Le verrou **exclusif** (*exclusive lock*) interdit la pose de tout autre verrou, exclusif ou partagé, et donc de toute lecture ou écriture par une autre transaction.

Verrous = **blocages**. Il faut en poser

- **le moins possible**, surtout pour les verrous exclusifs.
- **pour la durée la plus courte possible**

# Transactions et verrous

Le contrôleur pose des verrous pour une transaction $T$.

- On ne peut poser un **verrou partagé** sur un tuple $a$ que s'il n'y a que des verrous partagés sur ce tuple.
- On ne peut poser un **verrou exclusif** que si

  il n'y a aucun autre verrou, **ou**,
  il y a un verrou partagé déjà posé par $T$ elle-même (*upgrade*)

- Une transaction qui n'obtient pas un verrou, **est mise en attente**.
- Les verrous ne sont libérés qu'au moment du `commit` ou `rollback`.

# L'algorithme

On pose des verrous pour empêcher l'apparition de cycles.

- **Lecture** sur $x$, on regarde s'il y a un verrou **exclusif** sur $x$ ;
  si oui la transaction $T_i$ est mise en attente.
  si non, la transaction pose un verrou en lecture et l'opération est exécutée.
- **Ecriture** sur $x$, on regarde s'il y a un verrou **quelconque** sur $x$ ;
  si oui la transaction $T_i$ est mise en attente.
  si non, la transaction pose un verrou en écriture et l'opération est exécutée.

**Important** : ne fonctionne que si les verrous ne sont pas relâchés avant le `commit` ou le `rollback`.

# Exemple

Prenons l'exécution concurrente : $r_1[x]w_2[x]w_2[y]C_2w_1[y]C_1$

- $T_1$ pose un verrou partagé sur $x$, lit $x$ mais ne relâche pas le verrou ;
- $T_2$ tente de poser un verrou exclusif sur $x$ : impossible puisque $T_1$ détient un verrou partagé, *donc $T_2$ est mise en attente* ;
- $T_1$ pose un verrou exclusif sur $y$, modifie $y$, valide ; ses verrous sont relâchés ;
- $T_2$ est libérée : elle pose un verrou exclusif sur $x$, et le modifie ;
- $T_2$ pose un verrou exclusif sur $y$, et modifie $y$ ;
- $T_2$ valide, ce qui relâche les verrous sur $x$ et $y$.

Exécution après réordonnancement : $r_1[x]w_1[y]w_2[x]w_2[y]$

# Un gros problème : les `deadlock`

Reprenons notre exemple des mises à jour perdues.

$$r_1(s)r_1(c_1)r_2(s)r_2(c_2)w_2(s)w_2(c_2)C_2w_1(s)w_1(c_1)C_1$$

- $T_1$ lit $s$ et $c_1$, qui sont verrouillés en lecture.
- $T_2$ lit $s$ et $c_2$, qui sont verrouillés en lecture ; **$s$ partage deux verrous en lecture**
- $T_2$ veut écrire $s$ : conflit, donc blocage de $T_2$.
- $T_1$ veut écrire $s$ : conflit, donc blocage de $T_1$.

C'est **l'étreinte fatale** (*deadlock*) !!

Le système va rejeter une des transactions. Très regrettable,
mais encore mieux que d'introduire des anomalies ( ?)

# À retenir

Le 2PL est le seul algorithme actuellement utilisé pour garantir la sérialisabilité.

Il repose intensivement sur la pose de verrous qui ne sont relâchés qu'à la fin de la transaction.

Il entraîne des interblocages, et donc le rejet de certaines transactions : difficilement explicable pour un utilisateur, pose le problème de resoumettre la transaction.