

**Architecture d'un SGBD**  
**Transactions et Contrôle de la concurrence**  
**Marta Rukoz**

Introduction

Contrôle de concurrence

Gestion des accès concurrents et SQL

# introduction

1. Notion de transaction
2. Transactions concurrentes
3. Le problème des annulations

# introduction

## **Transaction : unité logique de travail**

action ou série d'opérations d'un utilisateur ou d'une application, qui accède(nt) ou modifie(nt) les données de la base, *transformant la base de données d'un état cohérent en un autre état cohérent*

# Exemple de transaction

## Virement bancaire **sans transaction**

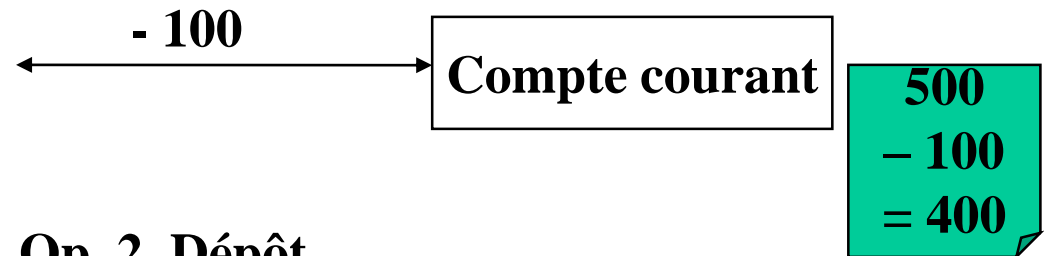
Virement =  
2 opérations atomiques

Que se passe-t-il si le  
Dépôt échoue ?

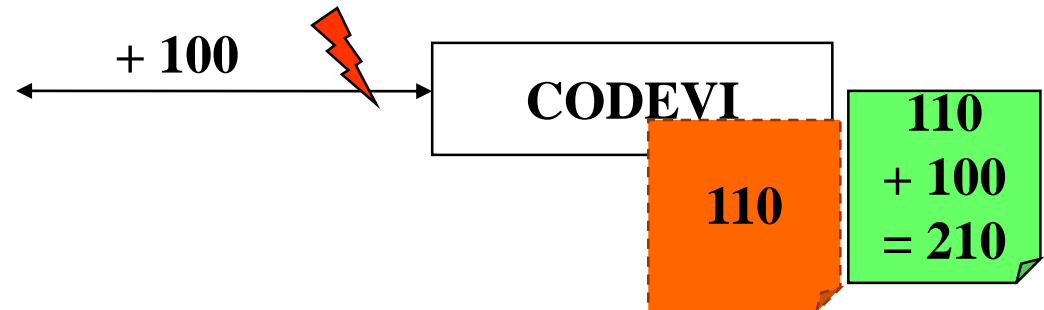
**Compte courant = 400**  
**CODEVI = 110**

**Appelez la banque !!!**

Op. 1. Retrait



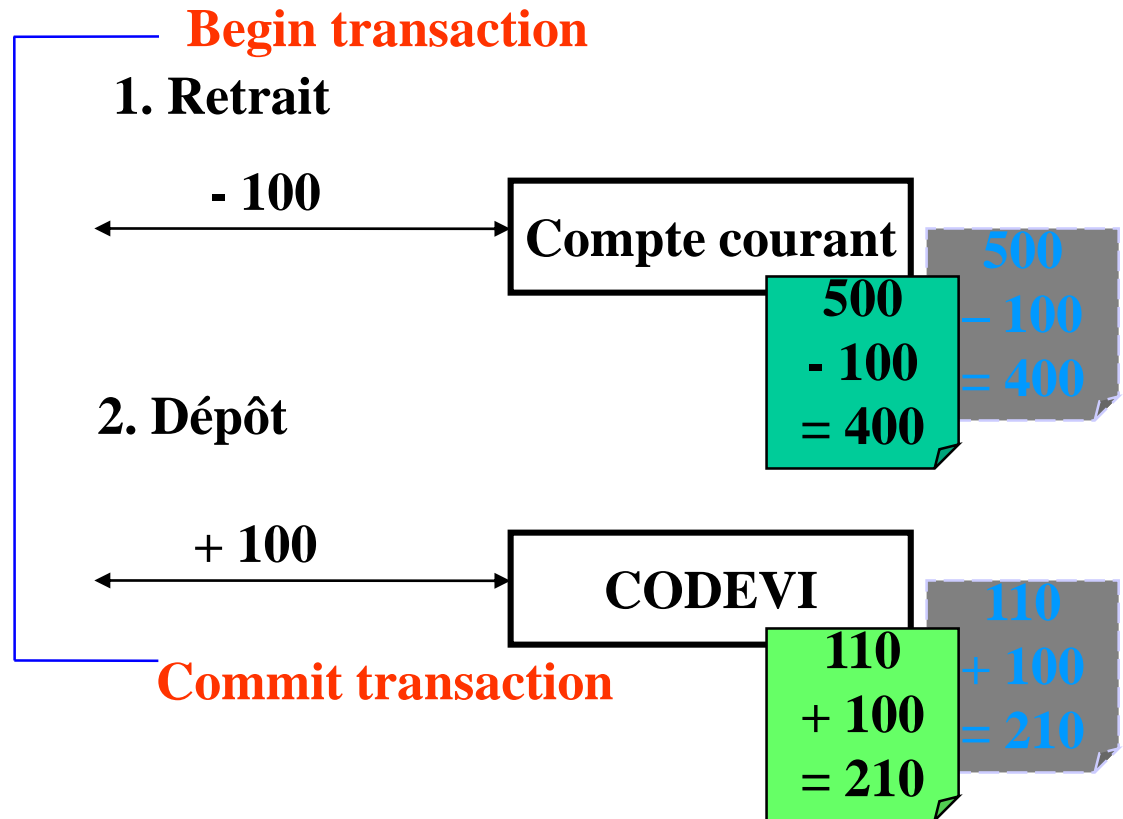
Op. 2. Dépôt



# Exemple de transaction

## Virement bancaire **dans une transaction** (1/2)

Virement = 1 transaction  
de 2 opérations  
atomiques



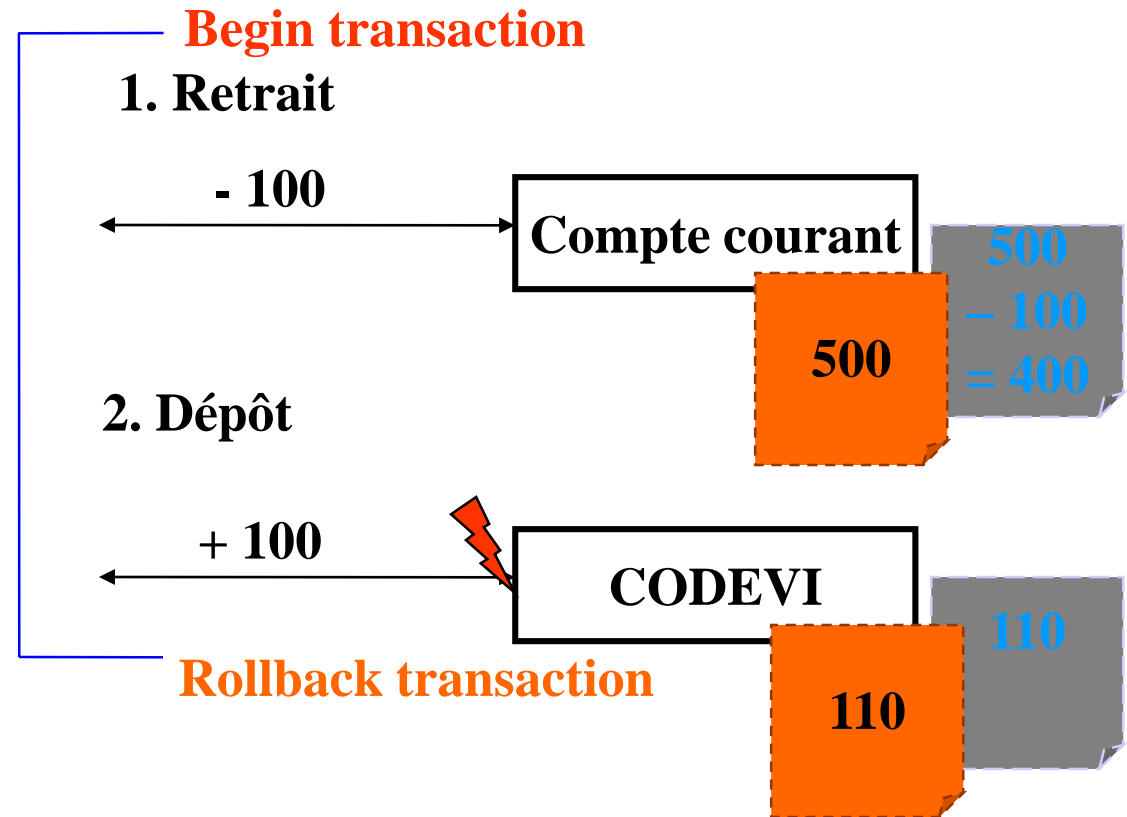
# Exemple de transaction

## Virement bancaire **dans une transaction** (2/2)

Que se passe-t-il si le  
Dépôt échoue ?

Compte courant = 500  
CODEVI = 110

Recommencez !



# Propriétés des transactions

- **Atomicité** : Tout ou rien

Une transaction effectue toutes ses opérations ou aucune.

En cas d'annulation, les modifications engagées doivent être défaites.

- **Cohérence** : Intégrité des données

Passage d'un état cohérent de la base à un autre état cohérent de la base de données

- **Isolation** : Pas d'interférence entre transactions

Les résultats d'une transaction ne sont visibles par les autres transactions qu'après sa validation

- **Durabilité** : Journalisation des mises à jour

Les modifications effectuées sont garanties même en cas de panne

# Gestionnaire de transactions (Moniteur des transactions ou moniteur TP)

Système qui assure une *gestion des transactions*. Il doit apporter la garantie que, lors de l'exécution d'une ou d'un ensemble de transactions, toutes les propriétés d'une transaction sont satisfaites, même en cas de défaillance du système



# Opérations des transactions

- Lecture
- Écriture
- Validation (*commit* en anglais). Elle consiste à rendre les mises à jour permanentes.
- Annulation (*rollback* en anglais). Elle annule les mises à jour effectuées.

Ainsi, une transaction est l'ensemble des instructions séparant un commit ou un rollback du commit ou du rollback suivant.

## **NOTATION :**

$ri[x]$  : la transaction  $T_i$  lit la valeur de  $x$

$wi[x]$  : la transaction  $T_i$  écrit la valeur de  $x$

$C_i$  : la transaction  $T_i$  valide (commit)

$R_i$  : la transaction  $T_i$  abandonne (Rollback)

# Opérations des transactions

Transaction = [**BEGIN TRANSACTION**]

**opérations de lectures et/ou écritures**

**COMMIT / ROLLBACK**

Exemple :

$T_1 : r_1[x] \ w_1[x] \ C_1$  (crédit)

$T_2 : r_2[x] \ w_2[x] \ r_2[y] \ w_2[y] \ C_2$  (transfert)

$T_3 : r_3[x] \ \mathbf{R}_3$  (transfert impossible)

$T_4 : r_4[x] \ r_4[y] \ w_4[x] \ \mathbf{R}_4 \ w_4[y] \ C_4$  ←

} correct

incorrect

# Exécution concurrent des transactions

- Par définition toute transaction  $T$  est correcte  $\Rightarrow$  à partir d'un état cohérent son exécution amène à un autre état cohérent.
- Pour des raisons d'efficacité, il est nécessaire d'autoriser l'exécution concurrente des transactions. Tout le problème réside alors dans le fait qu'il faut gérer ces exécutions concurrentes pour qu'elles ne conduisent pas à une base incohérente.
- C'est l'*entrelacement* non contrôlé des opérations de deux transactions correctes qui peut produire un résultat global incorrect.
- Les opérations sont toujours *exécutées en séquence*.

# Problèmes de concurrence

- Le problème des analyses incohérentes

$T_1$  = transfert de  $x \rightarrow y$  de 50

$T_2$  = calcul dans  $z$  de la somme  $x + y$

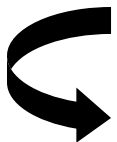
au début,  $x = 200$ ,  $y = 100$

$T_1 = r_1[x] \ w_1[x] \ r_1[y] \ w_1[y] \ C_1$

$T_2 = r_2[x] \ r_2[y] \ w_2[z] \ C_2$

$H = r_1[x]_{x=200} \ w_1[x]_{x=150} \ r_2[x]_{x=150} \ r_2[y]_{y=100} \ w_2[z]_{z=250} \ C_2 \ r_1[y]_{y=100} \ w_1[y]_{y=150} \ C_1$

Résultat :  $z = 250$  au lieu de  $z = 300$  ( $r_2[x]$  est influencé par  $T_1$  mais pas  $r_2[y]$ )



Problème de cohérence à cause de l'isolation.

# Problèmes de concurrence

- Le problème de perte de mise à jour

Exemple :

$T_1 = r_1[x] \ w_1[x] \ C_1$  (crédit x de 100)

$T_2 = r_2[x] \ w_2[x] \ C_2$  (crédit x de 50)

au début,  $x = 200$

Si  $H = r_1[x]_{x=200} \ r_2[x]_{x=200} \ w_1[x]_{x=300} \ w_2[x]_{x=250} \ C_1 \ C_2$

Résultat :  $x = 250$  au lieu de  $x = 350$  ( $w_1[x]$  est perdu à cause de  $w_2[x]$ )

 Problème de cohérence même si l'isolation est respectée.

# Problèmes de concurrence

- Le problème des dépendances non validées

Exemple :

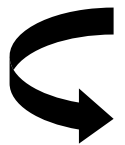
$T_1 = r_1 [x] \ w_1 [x] \ R_1$  (crédit x de 100)

$T_2 = r_2 [x] \ w_2 [x] \ C_2$  (crédit x de 50)

au début,  $x = 200$

Si  $H = r_1[x]_{x=200} \ w_1[x]_{x=300} \ r_2[x]_{x=300} \ w_2[x]_{x=350} \ C_2 \ R_1$

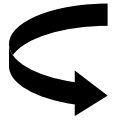
Résultat :  $x = 350$  au lieu de  $x = 250$  ( $T_1$  est annulée)

  $r_2[x]$  utilise la valeur non validée de x écrite par  $w_1 [x]$   
Problème de cohérence à cause de l'isolation.

# Le problème des annulations

- Annuler dans la base de données les effets d'une transaction T signifie :

- annuler les écritures de T
- annuler les transactions qui utilisent les écritures de T

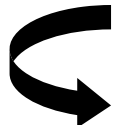


Donc, une transaction validée risque d'être annulée !

# Le problème des annulations

- Exemple 1:

$w_1[x] \ r_2[x] \ w_2[y] \ C_2 \ R_1$

  $R_1$  oblige l'annulation de  $T_2$ . Or, ce n'est pas possible car  $T_2$  a été validée.

- Solution : si  $T_2$  lit au moins un enregistrement dont  $T_1$  est la dernière transaction à l'avoir mis à jour, alors  $T_2$  doit valider après  $T_1 \rightarrow$  *retardement des validations*

Dans l'exemple : retardement de  $C_2$  après la fin de  $T_1$



# Annulations en cascade

- Dans l'exemple précédent, même si  $C_2$  est retardé,  $T_2$  sera quand même annulée à cause de  $T_1$

## Exemple 2

$w_1[x] \ r_2[x] \ w_2[y] \ R_1 \longrightarrow R_1 \text{ oblige } R_2$

- Solution :  $T_2$  ne doit lire qu'à partir de transactions validées  
→ *retardement des lectures*

Dans l'exemple : retardement de  $r_2[x]$  après la fin de  $T_1$

# Exécution stricte

- Exemple :

$w_1[x,2]$   $w_2[x,3]$   $R_1$   $R_2$

au début,  $x=1$

Image avant ( $w_1[x]$ ) = 1, Image avant ( $w_2[x]$ ) = 2

$R_1$  restaure  $x = 1$  : erreur     $R_2$  restaure  $x = 2$  : erreur

- Solution :  $w_2[x]$  attend que tout  $T_i$  qui a écrit  $x$  se termine (par  $R_1$  ou par  $C_1$ ), donc retardement de  $w_2[x]$  après la fin de  $T_1$ .

→ *retardement des lectures et des écritures*

# Le contrôle de concurrence

1. Théorie de la sérialisabilité
2. Contrôle par verrouillage à deux phases

# Théorie de la sérialisabilité

L'exécution en série d'un ensemble des transactions est toujours correcte.

En effet :

{ État initial }

T1

{État I1 }

T2

...

Tn

{État final }

Exécution concurrent (EC) équivalent à une exécution en série  
➡ (ES), alors EC est correcte

# Théorie de la sérialisabilité

## Exemple transactions :

$T_1$  = crédit de 100;

$r_1 [x]_{x = x1}$

$w_1 [x]_{x = x1+100}$

$c_1$

$T_2$  = crédit de 50;

$r_2 [x]_{x = x2}$

$w_2 [x]_{x = x2+50}$

$c_2$

## Exécution en série :

Au début  $x = 200$ .

$H1 = T1 \ T2 =$

$r1 [x]_{x = 200}$

$w1 [x]_{x = 300}$

$c1$

$r2 [x]_{x = 300}$

$w2 [x]_{x = 350}$

$c2$

$H2 = T2 \ T1 =$

$r2 [x]_{x = 200}$

$w2 [x]_{x = 250}$

$C2$

$r1 [x]_{x = 250}$

$w1 [x]_{x = 350}$

$C1$

# Théorie de la sérialisabilité

- Objectif : produire une exécution sérialisable des transactions, c'est-à-dire équivalente à une exécution en série quelconque des transactions.
- Equivalence de deux exécutions (histoires)
  - avoir les mêmes transactions et les mêmes opérations
  - produire le même effet sur la BD (écritures)
  - produire le même effet dans les transactions (lectures)

# Théorie de la sérialisabilité

## Opérations Conflictuel

Deux opérations  $O_1$  y  $O_2$  sont en conflit si :

- elles son produites par de transactions différentes
- réagissent sur la même variable
- L'une d'entre elles est une écriture

Exemple:

$(r_i[x] \ w_j[x])$  et  $(w_i[x] \ w_j[x])$  sont des opérations conflictuelles

# Théorie de la sérialisabilité

- Critère d'équivalence utilisé
  - avoir les mêmes transactions et les mêmes opérations
  - avoir le même ordre des opérations conflictuelles dans les transactions non annulées

• Exemple :  $T_1 : r_1[x] \ w_1[y] \ w_1[x] \ C_1$   
 $T_2 : w_2[x] \ r_2[y] \ w_2[y] \ C_2$

$H_1 : r_1[x] \ w_2[x] \ w_1[y] \ r_2[y] \ w_1[x] \ w_2[y] \ C_1 \ C_2$

Conflits :  $r_1[x]-w_2[x]$  ;  $w_2[x]-w_1[x]$  ;  $w_1[y]-r_2[y]$  ;  $w_1[y]-w_2[y]$

$H_2 : r_1[x] \ w_1[y] \ w_2[x] \ w_1[x] \ C_1 \ r_2[y] \ w_2[y] \ C_2$

Conflits :  $r_1[x]-w_2[x]$  ;  $w_2[x]-w_1[x]$  ;  $w_1[y]-r_2[y]$  ;  $w_1[y]-w_2[y]$

Donc  $H_2$  est équivalent à  $H_1$ .



# Théorie de la sérialisabilité

- Définition :

SG = graphe de sérialisation d'une exécution H :

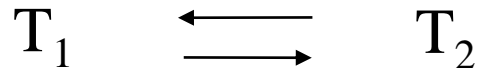
nœuds = transactions T validées dans H

Arcs : si  $O_1$  et  $O_2$  conflictuelles,  $O_1 \in T_i$ ,  $O_2 \in T_j$ ,

$O_1$  avant  $O_2 \Rightarrow$  arc  $T_i \rightarrow T_j$

H est sérialisable  $\Leftrightarrow$  SG(H) acyclique

- Exemple de graphe de sérialisation pour  $H_1$  et  $H_2$  :



$H_1$  et  $H_2$  ne sont donc pas sérialisables.

# Théorie de la sérialisabilité

## Condition suffisant pour l'équivalence

Deux histoires  $H1$  y  $H2$  sont équivalentes si pour toute couple des opérations conflictuelles  $O_i$  y  $O_j$ ,

Si  $O_i$  précède  $O_j$  dans  $H1 \Rightarrow O_i$  précède  $O_j$  dans  $H2$

# Contrôle par verrouillage à deux phases

- **Protocole**

- Avant d'agir sur un objet, une transaction doit obtenir un verrou sur cet objet.
- Après l'abandon d'un verrou, une transaction ne doit plus jamais pouvoir obtenir de verrous.

Une transaction obéissant à ce protocole a donc deux phases, une phase d'acquisition de verrous et une phase d'abandon de verrous.

Le respect du protocole est assuré par un module dit *scheduler* qui reçoit les opérations émises par les transactions et les traite.

- **Théorème**

Si toutes les transactions satisfont le protocole de verrouillage à deux phases, tous les ordonnancements entrelacés possibles sont sérialisables.

# Contrôle par verrouillage à deux phases

- **Algorithme**

1) Le scheduler reçoit  $O_i[x]$  et consulte le verrou déjà posé sur  $x$ ,  $ql_j[x]$ , s'il existe.

- si  $O_i[x]$  est en conflit avec  $ql_j[x]$ ,  $O_i[x]$  est retardée et la transaction  $T_i$  est mise en attente

- sinon  $T_i$  obtient le verrou  $O_i[x]$  et l'opération  $O_i[x]$  est exécutée

2) Un verrou pour  $ql_i[x]$  n'est jamais relâché avant la confirmation de l'exécution par le gestionnaire des données ( $C_i$ )

3) Dès que  $T_i$  relâche un verrou, elle ne peut plus en obtenir d'autre.

- **Théorème**

Toute exécution obtenue par un verrouillage à deux phases est sérialisable.

On obtient une exécution stricte en ne relâchant les verrous qu'au moment du commit ou du rollback. Les transactions obtenues satisfont les propriétés ACID.

# Contrôle par verrouillage à deux phases

- Exemple :

$T_1 : r_1 [x] w_1 [y] C_1$

$T_2 : r_2 [y] w_2 [y] C_2$

Ordre de réception :  $r_1 [x] r_2 [y] w_1 [y] C_1 w_2 [y] C_2$

$r_1 [x]$  exécutée

$r_2 [y]$  exécutée

$w_1 [y]$  retardée à cause de  $r_2 [y]$  et tout le reste de  $T_1$  va être bloqué

$C_1$  bloqué

$w_2 [y]$  exécutée

$C_2$  relâche les verrous sur  $y$

$w_1 [y]$  exécutée

$C_1$  exécutée

Exécution correcte :  $r_1 [x] r_2 [y] w_2 [y] C_2 w_1 [y] C_1$

Conflits :  $r_2 [y] - w_1 [y]$  ;  $w_2 [y] - w_1 [y]$

Pas de cycle, donc H sérialisable

# Contrôle par verrouillage à deux phases

- Cette stratégie peut parfois conduire à un *interblocage* (Un transaction T1 attend que une autre transaction T2 relâche un verrou et T2 attend aussi que T1 relâche un verrou)

## Exemple :

$T_1 : r_1 [x] w_1 [y] C_1$

$T_2 : w_2 [y] w_2 [x] C_2$

Ordre de réception :  $r_1 [x] w_2 [y] w_2 [x] w_1 [y]$

$T_1$  obtient un verrou pour  $r_1[x]$ ,  $T_2$  pour  $w_2[y]$

$w_2[x]$  attend  $r_1[x]$ ,  $w_1[y]$  attend  $w_2[y]$

interblocage de  $T_1$  et de  $T_2$ .

## Solutions :

- rejet de transactions non terminées après une durée limite
- annulation des transactions les moins coûteuses,
- ...

# Gestion des accès concurrents et SQL

- Il est possible en SQL de choisir explicitement le niveau de protection que l'on souhaite obtenir contre les incohérences résultant de la concurrence d'accès.
- Options possibles :
  - 1) On spécifie qu'une transaction ne fera que des lectures  
`SET TRANSACTION READ ONLY;`
  - 2) Une transaction peut lire et écrire (Option par défaut )  
`SET TRANSACTION READ WRITE;`

# SQL2 et les propriétés des exécutions concurrentes

- La norme SQL2 spécifie que ces exécutions doivent être sérialisables (mode par défaut). Un verrouillage strict doit alors être assuré par le SGBD.
- SQL2 propose des options moins fortes :

SET TRANSACTION ISOLATION LEVEL *option*

Liste des options :

1. READ UNCOMMITTED : on autorise les lectures de tuples écrits par d'autres transactions mais non encore validées
2. READ COMMITTED : on ne peut lire que les tuples validés (pas de verrou sur une donnée lue). C'est le mode par défaut d'ORACLE.