Transaction et concurrence

Du Mooc Bador
Serge Abiteboul, Benjamin Nguyen, Philippe Rigaux

C018SA-W1-S1

informatiques mathématiques

SEMAINE 1 : Transactions et concurrence

1. **Introduction : les transactions**
2. Les problèmes
3. Sérialisabilité
4. Estampillage
5. Verrouillage à 2 phases
6. Degrés d'isolation dans les SGBD
7. Verrouillage hiérarchique

Transactions bancaires

- Alice souhaite verser à Bob 100 euros.
- Le SGBD doit effectuer 2 opérations sur la table COMPTE :
  - Retirer 100 euros au compte d'Alice
  - Ajouter 100 euros au compte de Bob

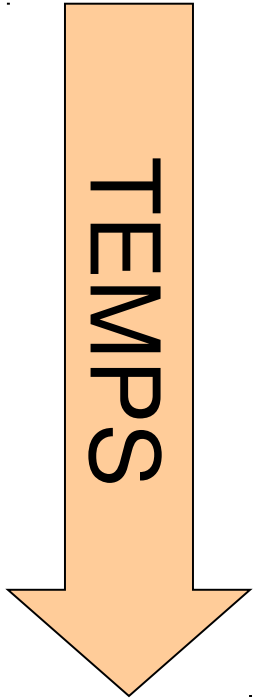
COMPTE

NC	CLIENT	SOLDE
0	Alice	1000
1	Bob	750
…	…	…

Transactions bancaires

t_1

UPDATE COMPTE
SET SOLDE = SOLDE – 100
WHERE NC = 0



COMPTE

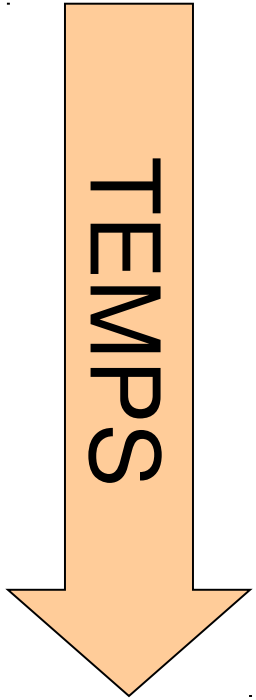
NC	CLIENT	SOLDE
0	Alice	1000
1	Bob	750
…	…	…

Somme : 1750

Transactions bancaires

t_1

```
UPDATE COMPTE
SET SOLDE = SOLDE – 100
WHERE NC = 0
```



COMPTE

NC	CLIENT	SOLDE
0	Alice	**900**
1	Bob	750
…	…	…

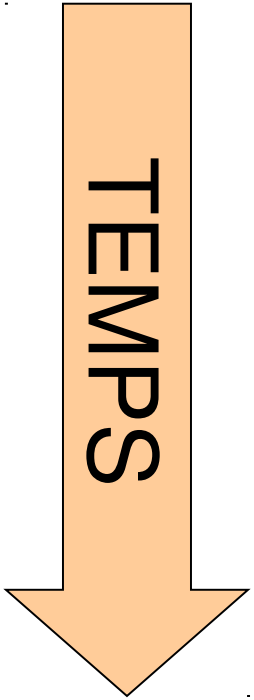
Somme : 1650

Transactions bancaires

t_1 UPDATE COMPTE
SET SOLDE = SOLDE – 100
WHERE NC = 0

t_2 UPDATE COMPTE
SET SOLDE = SOLDE + 100
WHERE NC = 1

t_3 **COMMIT** (valider) ou
ROLLBACK (avorter)



COMPTE

NC	CLIENT	SOLDE
0	Alice	900
1	Bob	**850**
…	…	…

Somme : 1750

Commit ou Rollback

- Les **deux** opérations (ou aucune)

doivent être validées pour maintenir

la cohérence des données : c'est **l'atomicité !**

- **COMMIT :** permet de valider **tous** les changements
- **ROLLBACK :** permet d'annuler **tous** les changements

On appelle **transaction** un ensemble séquentiel d'opérations permettant de passer d'un état cohérent à un autre.

COMPTE

NC	CLIENT	SOLDE
0	Alice	900
1	Bob	850
…	…	…

Concurrence

- Nombreuses transactions en parallèle
- Besoin pour le SGBD d'être capable de les gérer. Eviter :
  - Pertes d'opérations / introduction d'incohérences
  - Observation d'incohérences: Lectures non reproductibles / lectures fantômes

Des problèmes similaires apparaissent dans le cas des pannes.

Propriétés des transactions : ACIDité [Harder&Reuter, ACM CS 15(4), 1983]

- **Atomicité** : Toutes les MAJ sont exécutées ou aucune
- **Cohérence** : Passer d'un état cohérent à un autre (pas géré par le système)
- **Isolation** : La transaction s'effectue comme si elle était seule
- **Durabilité** : Une fois une transactions validée, son effet ne peut pas être perdu suite à une panne quelconque

SEMAINE 1 : Transactions et concurrence

1. **Introduction : les transactions**
2. Les problèmes
3. Sérialisabilité
4. Estampillage
5. Verrouillage à 2 phases
6. Degrés d'isolation dans les SGBD
7. Verrouillage hiérarchique

C018SA-W1-S2

SEMAINE 1 : Transactions et concurrence

1. Introduction : les transactions
2. **Les problèmes**
3. Sérialisabilité
4. Estampillage
5. Verrouillage à 2 phases
6. Degrés d'isolation dans les SGBD
7. Verrouillage hiérarchique

Problématique

1. Une base de données n’est pas interrogée et modifiée par un seul utilisateur.
2. Des problèmes d’incohérences peuvent apparaître lorsque plusieurs utilisateurs effectuent des opérations conflictuelles, ce qui peut être dû à un défaut d’isolation.

Problématique

1. Une base de données n’est pas interrogée et modifiée par un seul utilisateur.
2. Des problèmes d’incohérences peuvent apparaître lorsque plusieurs utilisateurs effectuent des opérations conflictuelles, ce qui peut être dû à un défaut d’isolation.
3. Quels sont ces incohérences ?
4. Quelles sont ces opérations ?
5. Comment éviter de se placer dans des situations d’opérations conflictuelles ?

Problématique

1. Des problèmes peuvent survenir lors de l'accès (lecture / écriture) concurrent sur des opérations successives
2. On aimerait que les opérations puissent se dérouler en isolation

> Nous allons étudier les aspects *isolation* des transactions.

3. Dans la suite : déterminer les opérations potentiellement conflictuelles

entre deux transactions.

Exemple de Base de Données

1. 1 table : EMP (NE, Nom, Sal)
2. 3 nuplets :

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Exemple de Base de Données

1. 1 table : EMP (NE, Nom, Sal)
2. 3 tuples :

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

… et deu eurs :

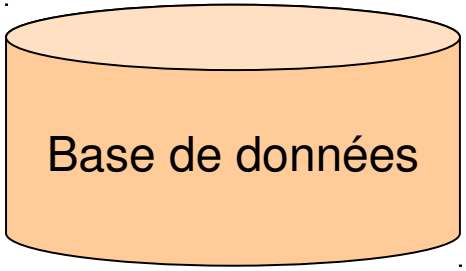
Alice e

Lecture de nuplets distincts

Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

Bob



NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

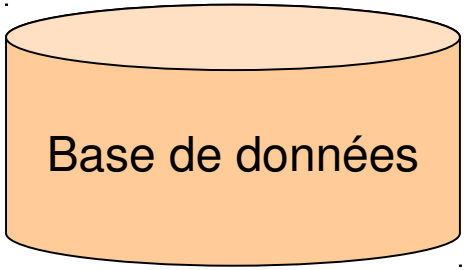
Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »



Bob

Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

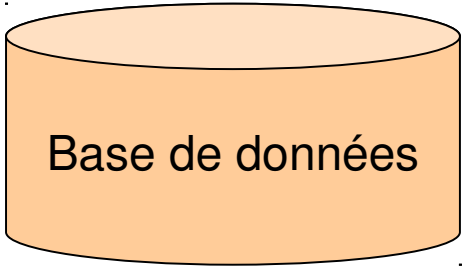
Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Diana »



Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Alice

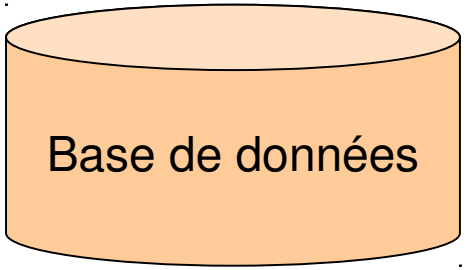
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Diana »

« 2100 »



Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Alice

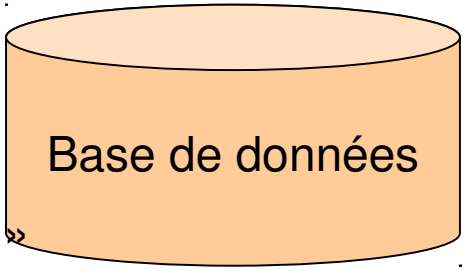
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Diana »

« 2100 »



Lecture de nuplets distincts

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Alice

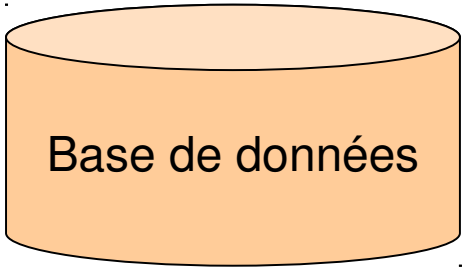
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Diana »

« 2100 »



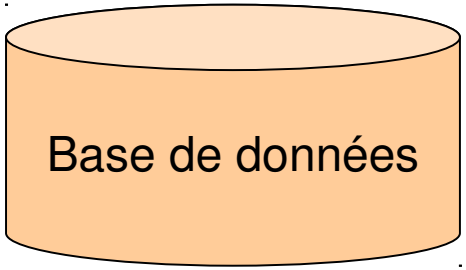
**PAS DE PROBLEME
SI A ET B LISENT DES
N-UPLETS DISTINCTS**

Lecture de nuplets identiques

Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

Bob



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

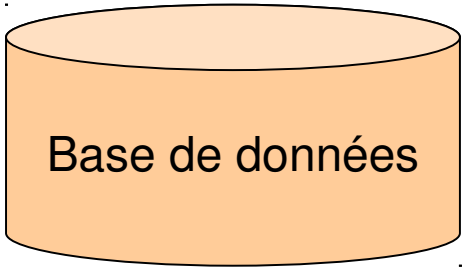
Lecture de nuplets identiques

Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture de nuplets identiques

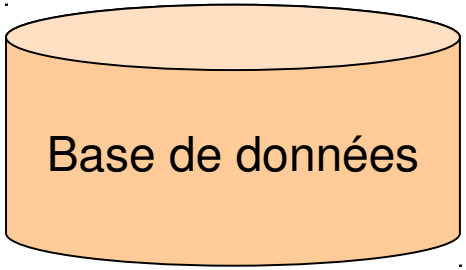
Alice

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Charlie »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture de nuplets identiques

Alice

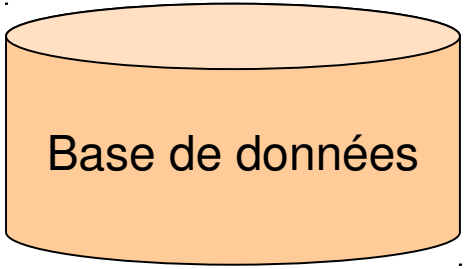
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Charlie »

« 2000 »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture de nuplets identiques

Alice

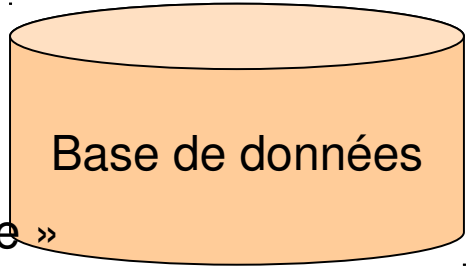
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture de nuplets identiques

Alice

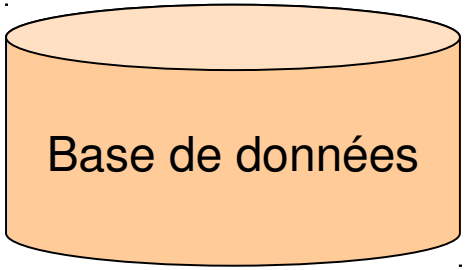
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »

« 2000 »

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM
= « Charlie »

« 2000 »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

PAS DE PROBLEME SI A ET B LISENT LE MEME N-UPLET

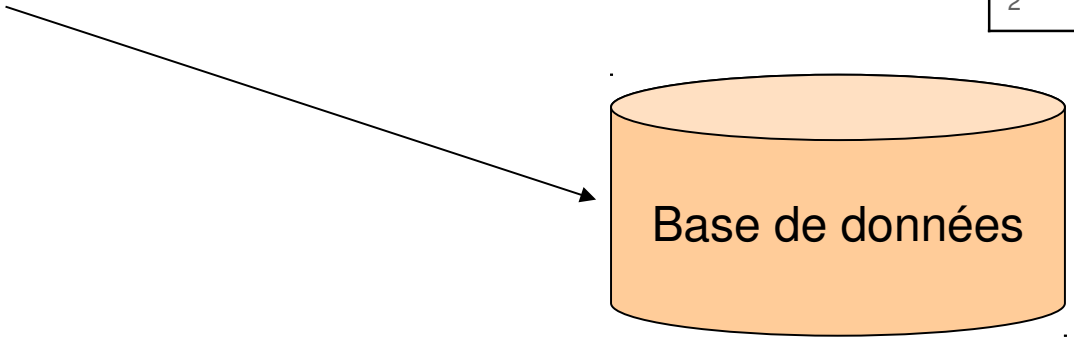
Lecture et écriture de nuplets différents

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

Bob

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600



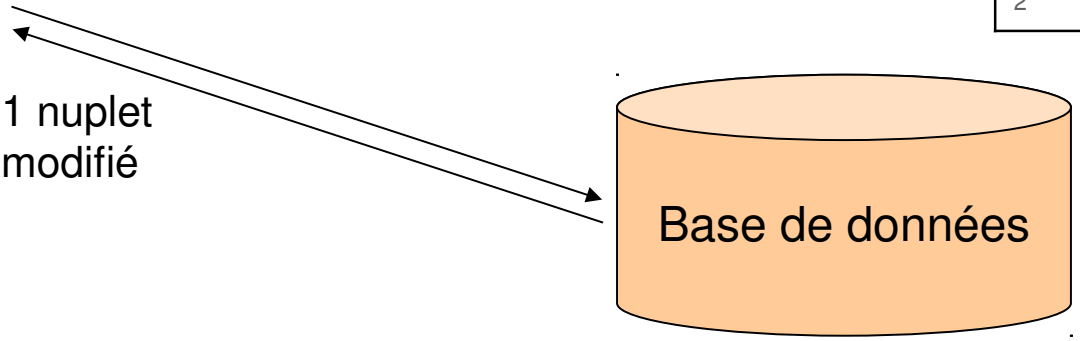
Lecture et écriture de nuplets différents

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob



NE	NOM	SAL
0	Charlie	**2050**
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets différents

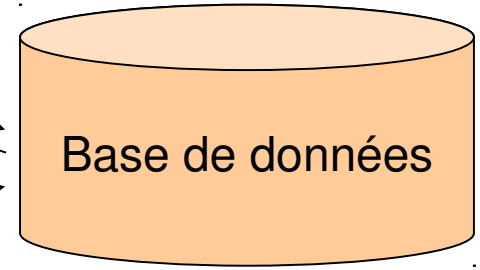
Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Diana »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets différents

Alice

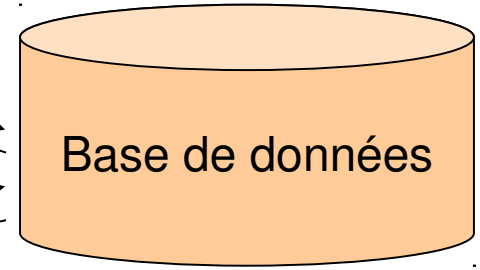
UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

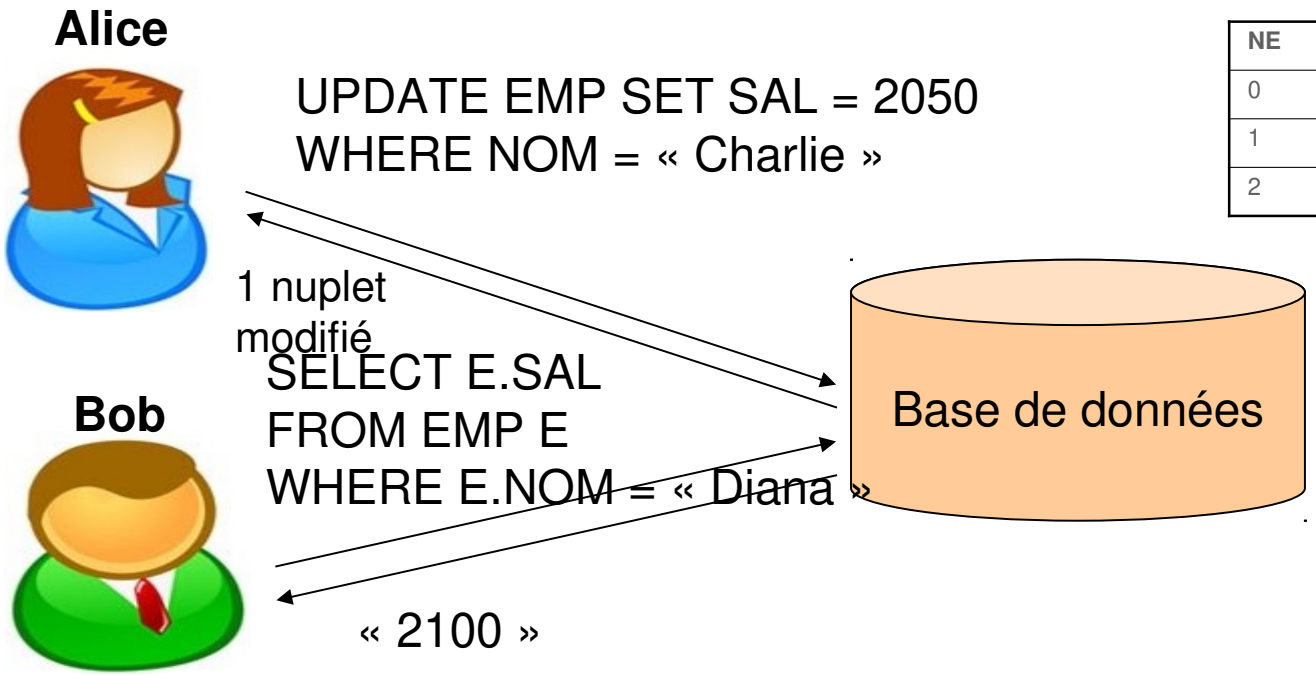
SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Diana »

« 2100 »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets différents



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets différents

Alice

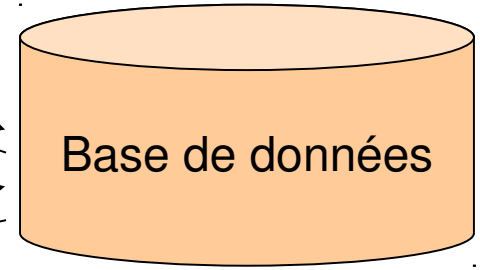
UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Diana »

« 2100 »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

PAS DE PROBLEME
SI A MODIFIE UN N-UPLET
QUE B NE LIT PAS
PAR LA SUITE

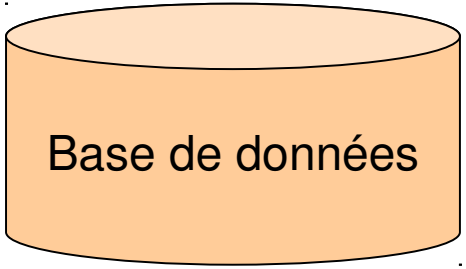
Lecture et écriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Bob



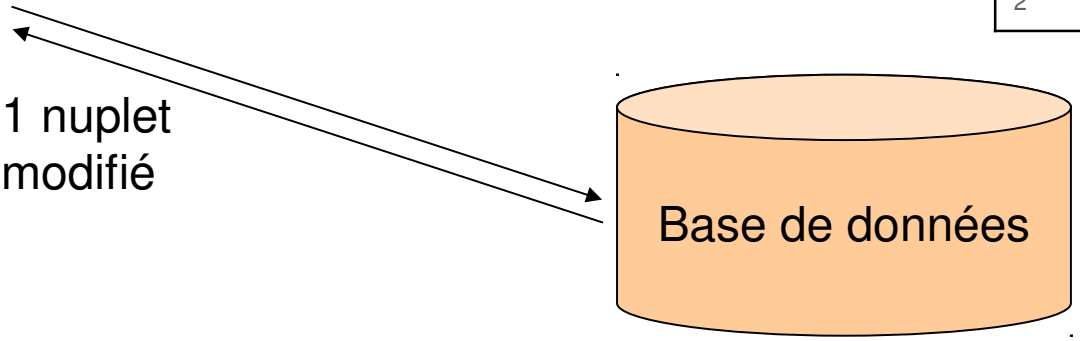
Lecture et écriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet
modifié

Bob



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets identiques

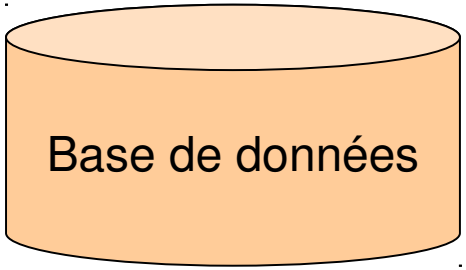
Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

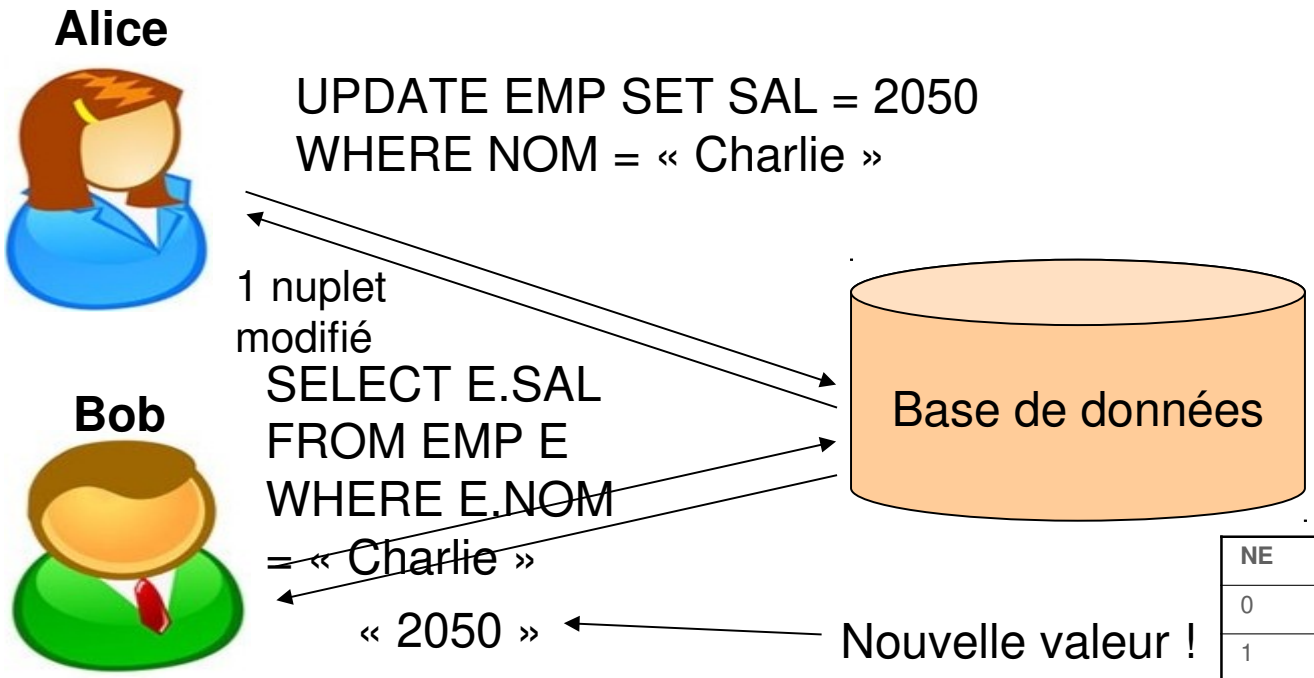
Bob

SELECT E.SAL
FROM EMP E
WHERE E.NOM = « Charlie »



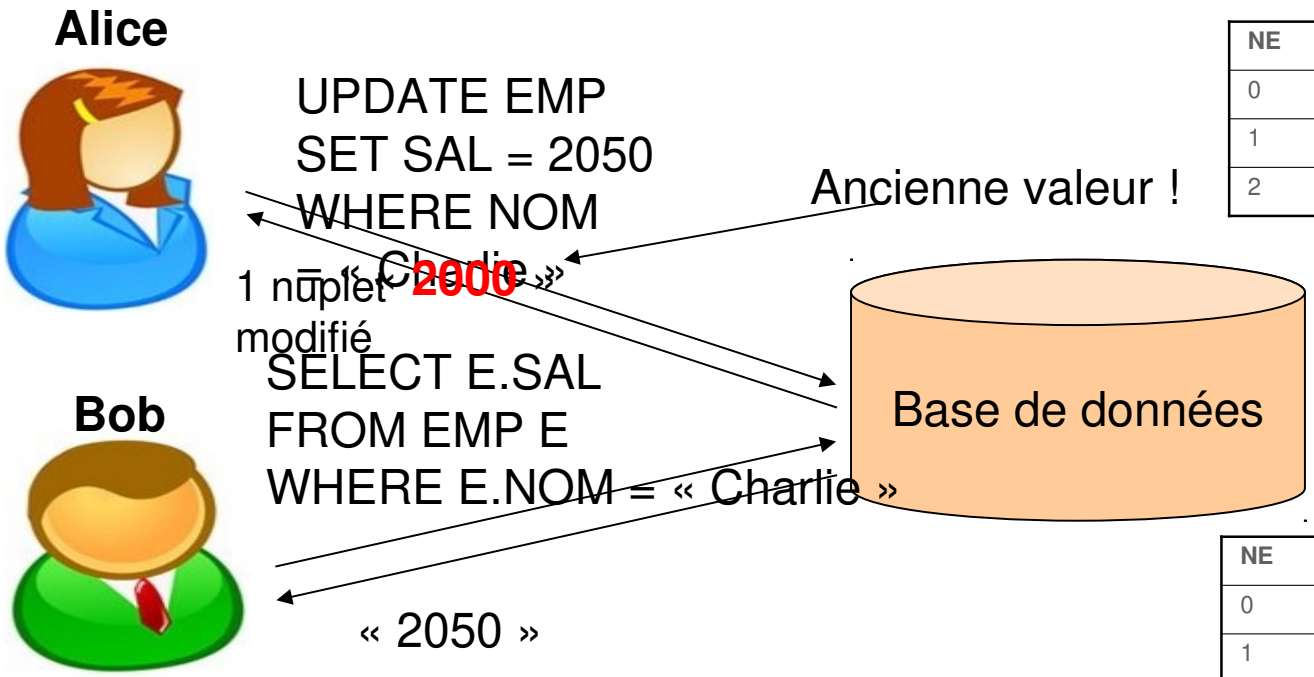
NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets identiques



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

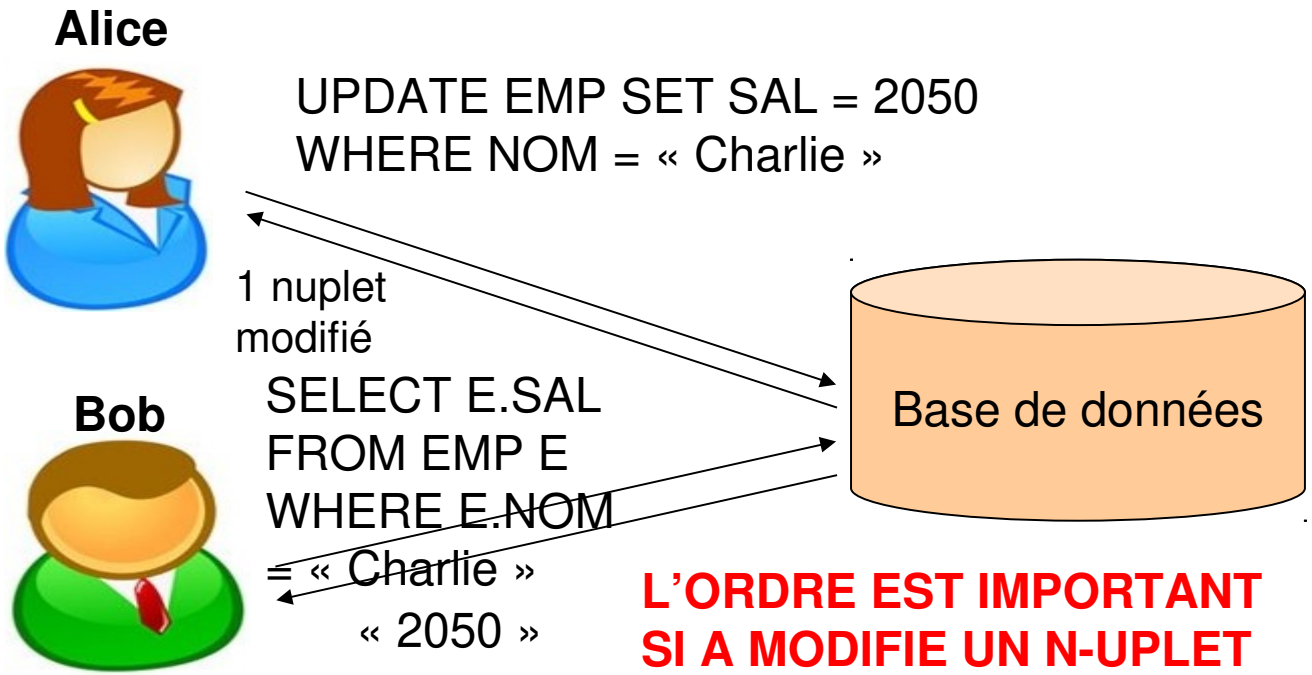
Lecture et écriture de nuplets identiques



NE	NOM	SAL
0	Charlie	**2000**
1	Diana	2100
2	Eric	1600

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Lecture et écriture de nuplets identiques



L'ORDRE EST IMPORTANT
SI A MODIFIE UN N-UPLET
QUE B A LU OU VA LIRE

Anomalie (problème) de la *lecture non reproductible*

Bob

Alice

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

T_1 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »
2000 ←

TEMPS

Anomalie (problème) de la *lecture non reproductible*

Bob

Alice

NE	NOM	SAL
0	Charlie	**2050**
1	Diana	2100
2	Eric	1600

T_1 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »
2000

TEMPS

T_2 UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

Anomalie (problème) de la *lecture non reproductible*

Bob

Alice

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

T_1 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »
2000 ←

TEMPS ↓

T_2 UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

T_3 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »
2050 ←

Anomalie (problème) de la *lecture non reproductible*

Bob

Alice

T_1 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »
2000

TEMPS

T_2 UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

B LIT DEUX FOIS LA MEME VALEUR ET OBTIENT DES RESULTATS DIFFERENTS !

T_3 SELECT E.SAL FROM EMP E
WHERE E.NOM = « Charlie »
2050

CETTE ANOMALIE EST DITE *LECTURE NON REPRODUCTIBLE.*

Nuplet lu de manière explicite ou implicite

1. Un nuplet peut être produit par la requête (explicite).
2. Un nuplet peut être utilisé pour produire le résultat de la requête (implicite) e.g. requêtes d'agrégats.
3. Une modification de l'un des nuplets utilisé pour calculer la requête causera donc un problème.

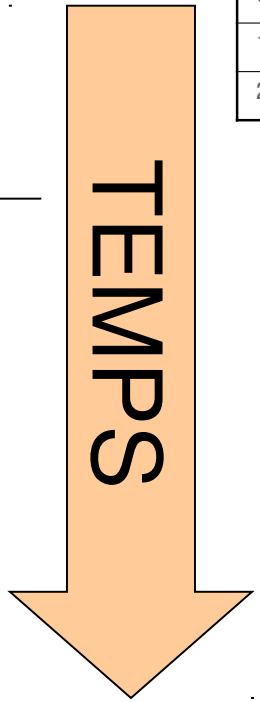
Anomalie (problème) de la *lecture fantôme*

Bob

Alice

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

T_1 SELECT AVG(E.SAL)
FROM EMP E
1900



Anomalie (problème) de la *lecture fantôme*

Bob

Alice

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600
3	**Flore**	**2300**

T_1 SELECT AVG(E.SAL)
FROM EMP E
1900

TEMPS

T_2 INSERT INTO EMP VALUES
(3, « Flore», 2300)

Anomalie (problème) de la *lecture fantôme*

Bob

Alice

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600
3	Flore	2300

T_1 SELECT AVG(E.SAL)
FROM EMP E
1900 ←

TEMPS

T_2 INSERT INTO EMP VALUES
(3, « Flore», 2300)

T_3 SELECT AVG(E.SAL)
FROM EMP E
2000 ←

B EXECUTE DEUX FOIS LA MEME REQUETE ET OBTIENT DES RESULTATS DIFFERENTS !

CETTE ANOMALIE EST DITE *LECTURE FANTOME.*

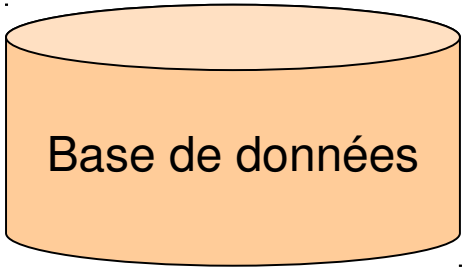
Ecriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

NE	NOM	SAL
0	Charlie	2000
1	Diana	2100
2	Eric	1600

Bob



Ecriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet
modifié

Bob

Base de données

NE	NOM	SAL
0	Charlie	**2050**
1	Diana	2100
2	Eric	1600

Ecriture de nuplets identiques

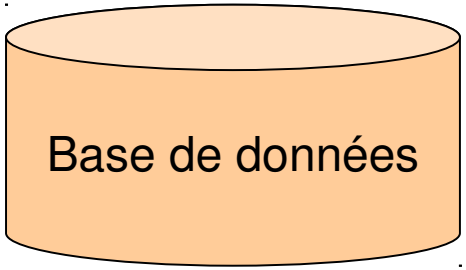
Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

UPDATE EMP
SET SAL = 3000
WHERE
NOM = « Charlie »



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

Ecriture de nuplets identiques

Alice

UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

UPDATE EMP
SET SAL = 3000
WHERE
NOM = « Charlie »

1 nuplet modifié

Base de données

NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600

NE	NOM	SAL
0	Charlie	**3000**
1	Diana	2100
2	Eric	1600

Ecriture de nuplets identiques

Alice

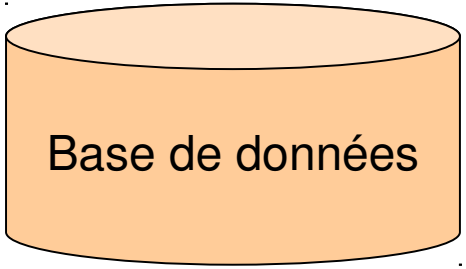
UPDATE EMP SET SAL = 2050
WHERE NOM = « Charlie »

1 nuplet modifié

Bob

UPDATE EMP
SET SAL = 3000
WHERE
NOM = « Charlie »

1 nuplet modifié



NE	NOM	SAL
0	Charlie	**3000**
1	Diana	2100
2	Eric	1600

SI A MODIFIE UN N-UPLET
PUIS B LE MODIFIE AUSSI
ALORS A PERD
SA MODIFICATION EN COURS
DE TRANSACTION

C018SA-W1-S3

SEMAINE 1 : Transactions et concurrence

1. Introduction : les transactions
2. Les problèmes
3. **Sérialisabilité**
4. Estampillage
5. Verrouillage à 2 phases
6. Degrés d'isolation dans les SGBD
7. Verrouillage hiérarchique

Objectifs du contrôle de concurrence

Transaction = séquence d'opérations

Correct : Un ordonnancement sériel
des transactions
exécuter toute la transaction T_1 puis T_2 etc… !

Correct : Un ordonnancement sérialisable
(à définir)
gère l'enchevêtrement des transactions

Il faut permettre :

- La meilleure fluidité possible
- La meilleure performance possible
- D'éviter les blocages

Formalisation : opérations de lecture et d'écriture

1. On travaille au grain du nuplet, noté x_i
   - x_0 = (0, « Alice », 1000), x_1 = (1, « Bob », 750) et x_2 = (2, « Charlie », 100)

- Une requête lisant une valeur d'un nuplet donné x_i correspond à une opération de **lecture** sur ce nuplet, notée **$R(x_i)$**.
  1. **SELECT SOLDE FROM COMPTE WHERE NC = 0** est une opération de lecture de x_0, notée **$R(x_0)$**
- Une requête modifiant une valeur d'un nuplet donné x_i correspond à une opération d'**écriture** sur ce nuplet, notée **$W(x_i)$**.
  1. **UPDATE COMPTE SET SOLDE = SOLDE + 100 WHERE NC = 1** est une opération d'écriture de x_1, notée **$W(x_1)$**

Opérations conflictuelles entre transactions

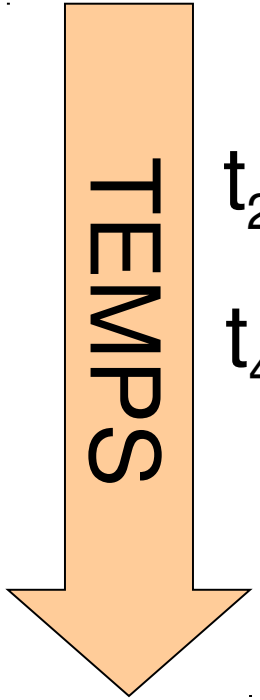
1. Deux opérations a et a’, exécutées respectivement par deux transactions différentes T et T' sont dites **conflictuelles** si l'exécution des deux séquences
   1. a;a’ (a puis a’) et
   2. a’;a (a’ puis a)
2. est *susceptible* de conduire

   à des résultats différents.

Exemple d'opérations conflictuelles

T_1 : transférer le solde de Bob à Alice

T_2 : transférer le solde de Charlie à Alice



t_1 SELECT SOLDE INTO S1
WHERE NC = 1

t_2 SELECT SOLDE INTO S2
WHERE NC = 2

t_3 UPDATE COMPTE
SET SOLDE = S1
WHERE NC = 0

…

t_4 UPDATE COMPTE
SET SOLDE = S2
WHERE NC = 0

…

Opérations conflictuelles

	R(X)	W(X)
R(X)	NON	OUI
W(X)	OUI	OUI

Transactions sérielles =
exécuter toute la transaction T_1 puis T_2 etc…

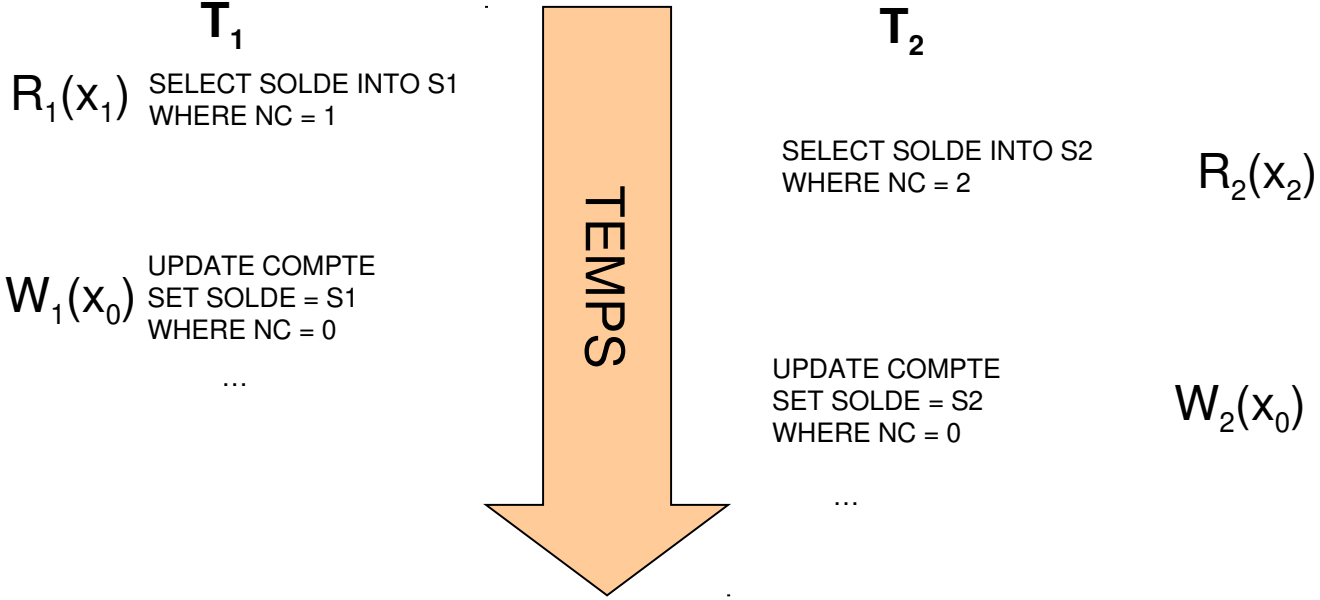
Pour garantir l'isolation des transactions, il suffit de pouvoir exécuter intégralement une transaction, puis une autre, etc.

Comment accepter plus de parallélisme ?

Solution : avoir un **ordonnancement sérialisable** des transactions
⇔ équivalent à des transactions sérielles

Ordonnancement

Définition : Un *ordonnancement* d'opérations de plusieurs transactions est composé d'opérations atomiques de type R ou W sur des données.



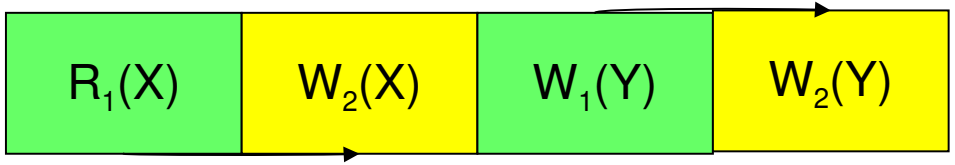
$$\Omega = R_1(x_1);R_2(x_2);W_1(x_0);W_2(x_0)$$

Ordonnancement

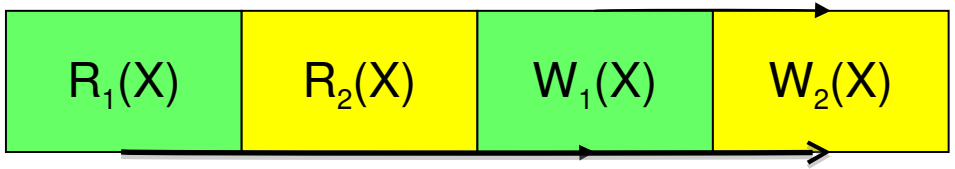
1. **Définition :** Un ordonnancement A est *équivalent* à un ordonnancement A' **ssi** le résultat produit et observé est le même.
2. **Proposition : Si** A' est un ordonnancement produit à partir d'un ordonnancement A en ne permutant que des opérations non conflictuelles **alors** A et A' sont équivalentes.

Sérialisabilité : Définition Informelle

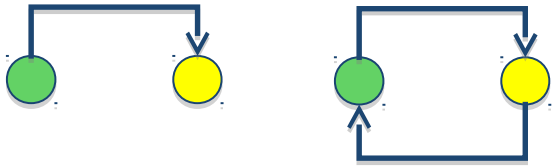
Un ordonnancement (d'opérations atomiques) est **sérialisable** si et seulement si tous les conflits sont « dans le même sens »



SERIALISABLE



PROBLEME



Sérialisabilité : Définition équivalente

Un ordonnancement est sérialisable

si et seulement si :

Il peut être transformé en un ordonnancement sériel par permutations successives d'opérations ne constituant pas une paire d'opérations conflictuelles.

Une telle transformation, si elle est possible, fournit effectivement un ordonnancement sériel équivalent.

Cyclicité du graphe de précédence

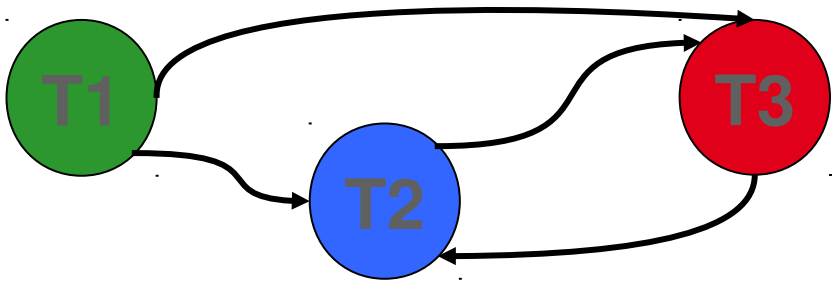
Graphe de précédence : un arc de Ti à Tj s'il y a dans l'ordonnancement une opération dans Ti avant une opération conflictuelle dans Tj

Proposition : l'ordonnancement est sérialisable si et seulement si son graphe de précédence est acyclique

Algorithme : maintenir le graphe de précédence et empêcher la création de cycle

Cyclicité du graphe de précédence

- $R_1(Y)$ $R_2(Y)$ $W_3(Y)$ $W_2(Y)$



Problème
- Gestion d'un gros graphe
- Détection de cycle dans ce graphe
- Que faire si le graphe est cyclique ?

Deux techniques pour garantir la sérialisabilité

1. Protocole d'estampillage : **stratégie curative**
2. Protocole de verrouillage : **stratégie préventive**

C018SA-W1-S4

SEMAINE 1 : Transactions et concurrence

1. Introduction : les transactions
2. Les problèmes
3. Sérialisabilité
4. **Estampillage**
5. Verrouillage à 2 phases
6. Degrés d'isolation dans les SGBD
7. Verrouillage hiérarchique

Estampillage : principe

On associe à chaque transaction T_i un numéro distinct, appelé **estampille**, qu'on note $E(T_i)$

Ce numéro est donné de manière croissante aux transactions selon leur date de début:

$$E(T_i) < E(T_j) \Leftrightarrow T_i \text{ a débuté avant } T_j$$

Estampillage : principe

Chaque nuplet x_i va mémoriser :

- L'estampille de la lectrice la plus récente, notée $E_R(x_i)$
- L'estampille de l'écrivaine la plus récente, notée $E_W(x_i)$

Règle (estampillage partiel) : Seule une transaction T_j t.q.

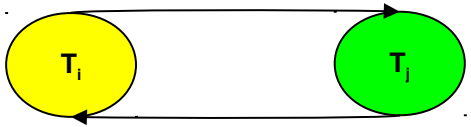
1. $E(T_j)>E_W(x_i)$ pourra accéder en lecture (→ MAJ de $E_R(X_i)$)
2. $E(T_j)>E_W(x_i)$ et $E(T_j)>E_R(x_i)$ en écriture (→ MAJ de $E_W(X_i)$)

Théorème : Tout ordonnancement respectant la règle d'estampillage partiel est sérialisable.

Preuve de la propriété de sérialisabilité

T_i a accédé à x_k puis T_j a accédé à x_k
→ $j > i$

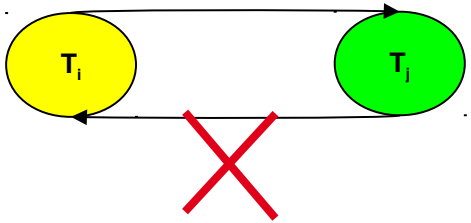
Preuve de la propriété de sérialisabilité



T_i a accédé à x_k puis T_j a accédé à x_k
→ $j > i$

T_j a accédé à x_k puis T_i a accédé à x_k
→ $i < j$

Preuve de la propriété de sérialisabilité



T_i a accédé à x_k puis T_j a accédé à x_k
→ $j > i$

T_j a accédé à x_k puis T_i a accédé à x_k
→ $i < j$

IMPOSSIBLE PAR CONSTRUCTION

Le graphe de précédence est un DAG par construction !
D'après la définition 3 cela équivaut au fait que l'ordonnancement soit sérialisable.

Une transaction viole la règle ?

Si une transaction T_j telle que $E(T_j) < E_w(x_i)$ tente d'écrire une nouvelle valeur dans x_i → T_j est annulée !

- On doit avorter T_j
- On doit défaire toutes les opérations d'écriture de T_j
- Toutes les transactions ayant lu des écritures de T_j sont aussi avortées
- Avortement en cascade ☹

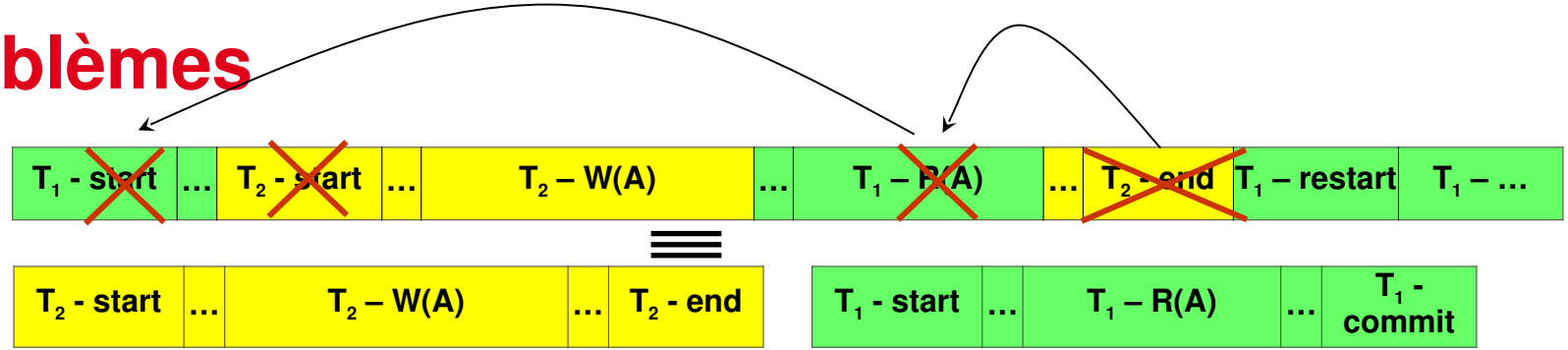
Bilan de l’estampillage

1. Algorithme simple : méthode préventive
2. Méthode « équitable »

→ on prend les transactions dans l’ordre où elles arrivent, en cas de problème, on annule

Vraiment équitable ?

Problèmes



1. Abandon en cascade
2. Abandon inutile (e.g. dans le cas de la lecture)
3. Risque de famine pour les transactions longues

C018SA-W1-S5

SEMAINE 1 : Transactions et concurrence

1. Introduction : les transactions
2. Les problèmes
3. Sérialisabilité
4. Estampillage
5. **Verrouillage à 2 phases (2PL)**
6. Degrés d'isolation dans les SGBD
7. Verrouillage hiérarchique

Le verrouillage

1. Estampillage : soit tout se passe bien, soit très mal
2. Fonctionne-t-on comme ça « dans la vraie vie » ?
   Files d'attente, trains, etc …

1. Garantir le bon fonctionnement

→ On pose des **verrous** de plusieurs types pour empêcher une utilisation potentiellement conflictuelle des ressources (nuplets)

→ Selon le type de verrou, d'autres
transactions pourront ou non interagir
avec la ressource

Le verrouillage à deux phases

- Une transaction se déroule en deux phases
  - Phase I : on a le droit de poser des verrous mais pas d'en libérer
  - Phase II : on a le droit de libérer des verrous mais pas d'en poser
- En d'autres termes : quand on a libéré un premier verrou, on n'a plus le droit d'en poser
- En pratique
  - On ne sait pas de quelles données, on aura besoin
  - On pose des verrous et en fin de transaction, on les libère tous

Principe de fonctionnement 2-Phase Locking

nuplet	Verrou
A	
B	

Verrous

1. Lecture : Verrou « S » (Shared)
2. Ecriture : Verrou « X » (Exclusive)

3. Tenter d'exécuter une lecture (SELECT) ou une écriture (UPDATE)

Verrou demandé \ Verrou détenu	S	X
S	Accordé	Mise en attente
X	Mise en attente	Mise en attente

ou

Principe de fonctionnement 2-Phase Locking

Phase 1 : pause de verrous

Phase 2 : enlève les verrous

nuplet	Verrou
A	S_1 X X2
B	S_2

$R_1(A)$	$R_2(B)$	$W_1(A)$	$COMMIT_1$	$W_2(A)$

$W_2(A)$ (en attente)

Verrou demandé \ Verrou détenu	S	X
S	Accordé	Mise en attente
X	Mise en attente	Mise en attente

Théorème : un ordonnancement construit avec 2PL est sérialisable

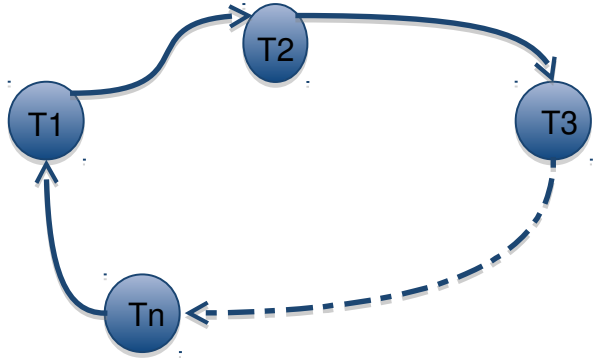
Preuve

- Supposons que l'ordonnancement ne soit pas sérialisable
- Il y a un cycle dans le graphe de précédence
- O1(A) … O2(A)

  T1 a libéré avant que T2 verrouille

  Phase 1 de T1 était finie avant que la phase 2 de T2 commence

  débutPhase2(T1) < débutPhase2(T2)



débutPhase2(T1) < débutPhase2(T2) < … < débutPhase2(T1)

CONTRADICTION

Problèmes du verrouillage à 2 phases

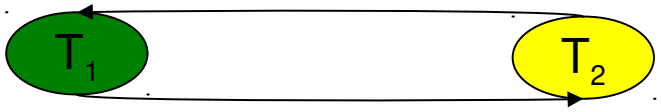
1. **Verrou mortel** (cycle) ou « Deadlock »
   - Transactions qui s‘attendent mutuellement

2. **Performance**
   - Petit granule (nuplet) : → verrouillage coûteux
   - Gros granule (page, table) → temps d‘attente plus important

Problèmes : verrou mortel

$R_1(A)$ $R_2(B)$ $W_1(B)$ $W_2(A)$

DEADLOCK !



Détection : graphe des attentes

- (comme le graphe de précédence)
- T_i attend T_j si T_i demande un verrou détenu par T_j
- Si un cycle apparaît, on a un verrou mortel !

Plus acceptable : moins d'arcs

Alternative : time-out (faux positifs)

Solutions aux deadlocks

Wait-Die : Technique non-préemptive (n'interrompt pas)

- Si T_i demande un verrouillage accordé à T_j et T_i est **plus vieille** alors T_i est mise en attente
- Si T_i demande un verrouillage accordé à T_j et T_i est **plus jeune** alors T_i est annulée

Wound-Wait : Technique préemptive (interrompt)

- Si T_i demande un verrouillage accordé à T_j et T_i est **plus vieille** alors T_i est annulée
- Si T_i demande un verrouillage accordé à T_j et T_i est **plus jeune** alors T_i est mise en attente

Résolution du problème de performances

- Demander la sérialisabilité *coûte cher* en terme de performances du SGBD
  - beaucoup d'opérations seront mises en attente.
  - Poser des verrous coûte cher
- Si le SGBD attend *sans rien faire* une transaction pour poursuivre c'est un gros problème !

  →on pourrait tolérer certaines anomalies : **Degrés d'isolation**

  →on pourrait verrouiller autrement : **Verrouillage hiérarchique**

C018SA-W1-S6

SEMAINE 1 : Transactions et concurrence

1. Introduction : les transactions
2. Les problèmes
3. Sérialisabilité
4. Estampillage
5. Verrouillage à 2 phases
6. **Degrés d'isolation dans les SGBD**
7. Verrouillage hiérarchique

Degrés d'isolation : la sérialisabilité coûte trop cher

1. Autoriser des transactions à transgresser la sérialisabilité → accroissement du parallélisme
2. Standardisés par SQL2 (Set Transaction Isolation Level)

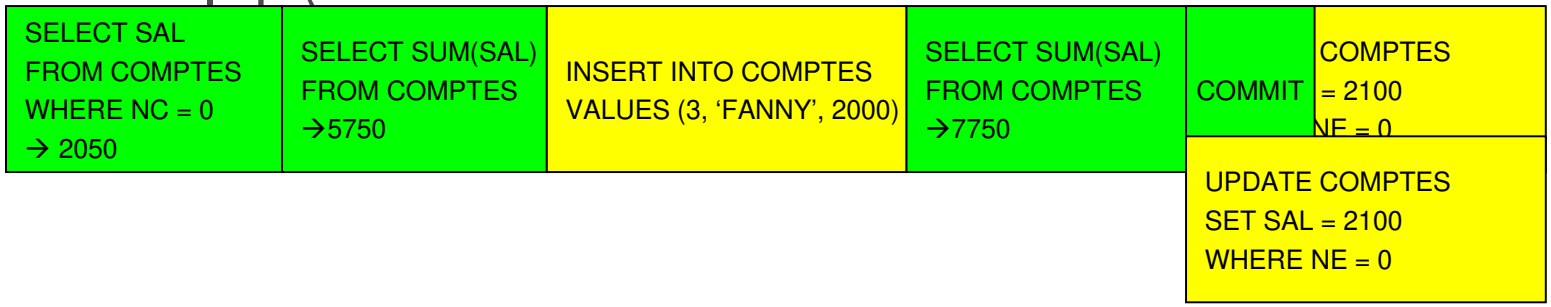
Niveau sérialisable
Niveau lectures reproductibles
Niveau lectures valides (propres)
Niveau lectures dégradées

Niveau Sérialisable (SERIALIZABLE)

1. La sérialisabilité complète des transactions est assurée
2. Les données auxquelles on accède sont verrouillées jusqu'à la fin de la transaction

Lectures reproductibles (REPEATABLE READ)

1. Lectures reproductibles, mais requêtes non reproductibles
2. Apparition de fantômes (Phantom Read = nouveau



NE	NOM	SAL
0	Charlie	2050
1	Diana	2100
2	Eric	1600
3	**Fanny**	**2000**

Lectures propres (READ COMMITTED)

1. Lectures et requêtes non reproductibles
2. Apparition de fantômes et changement de valeurs déjà lues

SELECT SAL FROM COMPTES WHERE NC = 0 → 2050	SELECT SUM(SAL) FROM COMPTES →5750	UPDATE COMPTES SET SAL = 2100 WHERE NE = 0	COMMIT	SELECT SUM(SAL) FROM COMPTES →5800	SELECT SAL FROM COMPTES WHERE NC = 0 → 2100

NE	NOM	SAL
0	Charlie	2100 ?
1	Diana	2100
2	Eric	1600

Lectures dégradées (READ UNCOMMITTED)

1. Lectures et requêtes non reproductibles
2. Lectures sales (« Dirty Read » = lectures non reproductibles, même sur des données non « commit ») : Lectures de données intermédiaires

SELECT SAL FROM COMPTES WHERE NC = 0 → 2050	SELECT SUM(SAL) FROM COMPTES →5750	UPDATE COMPTES SET SAL = 2100 WHERE NE = 0	SELECT SUM(SAL) FROM COMPTES →5800	SELECT SAL FROM COMPTES WHERE NC = 0 → 2100	ROLLBACK	SELECT SAL FROM COMPTES WHERE NC = 0 → 2050

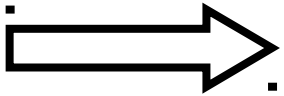
NE	NOM	SAL
0	Charlie	2050 ?
1	Diana	2100
2	Eric	1600

Bilan

Niveau d'Isolation	Lectures Sales	Lectures non reproductibles	Lectures fantômes
Read Uncommitted	Possibles	Possibles	Possibles
Read Committed		Possibles	Possibles
Repeatable Read			Possibles
Serializable			

Choix du niveau d'isolation

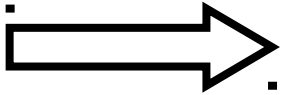
1. Beaucoup de lectures
2. Peu ou pas d'écritures
3. Transactions longues
4. Peu de transactions



`READ COMMITTED`

Mode par défaut Oracle

5. Peu de lectures
6. Peu d'écritures
7. Transaction courtes
8. Beaucoup de transactions

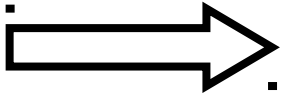


`SERIALIZABLE`

`REPEATABLE READ`

Mode par défaut MySQL

9. Systèmes d'inspection des données (débug)



`DIRTY READ`

C018SA-W1-S7

SEMAINE 1 : Transactions et concurrence

1. Introduction : les transactions
2. Les problèmes
3. Sérialisabilité
4. Estampillage
5. Verrouillage à 2 phases
6. Degrés d'isolation dans les SGBD
7. **Verrouillage hiérarchique**

Verrouiller : oui mais quoi ?

1. Verrouiller coût cher : **O(*n*)** où *n* est le nombre de nuplets de la table
2. **Idée :** verrouiller juste la table → **O(1)**

3. **Avantage :** Adapter la taille du verrou au nombre de transactions
    1. Granule gros → peu de verrous, temps d'attente long
    2. Granule fin → beaucoup de verrous, peu de temps d'attente

    3. Le bon compromis dépend du nombre de transactions en cours
    4. Deux niveaux de verrous : la table et le nuplet

Se généralise en **verrouillage hiérarchique**

Verrouiller : oui mais quoi ?

1. Verrouiller coût cher : **O(*n*)** où *n* est le nombre de nuplets de la table
2. **Idée :** verrouiller juste la table → **O(1)**

3. **Avantage :** Adapter la taille du verrou au nombre de transactions
    1. Granule gros → peu de verrous, temps d'attente long
    2. Granule fin → beaucoup de verrous, peu de temps d'attente

    3. Le bon compromis dépend du nombre de transactions en cours
    4. Deux niveaux de verrous : la table et le nuplet

Se généralise en **verrouillage hiérarchique**

Verrou table vs verrou nuplet

Même comportement des verrous :

	R(A)	W(A)
R(A)	NON	OUI
W(A)	OUI	OUI

1) verrous ligne

2) verrou table

EMP		Verrou : X	
NE	**NOM**	**SAL**	**Verrou ligne**
0	Charlie	2000 2100	X
1	Diana	2200 2300	X
2	Eric	1700 1800	X

UPDATE EMP SET SAL = SAL + 100

Verrou table vs verrou nuplet

Même comportement des verrous :

	R(A)	W(A)
R(A)	NON	OUI
W(A)	OUI	OUI

1) verrous ligne

2) verrou table

EMP		Verrou : X	
NE	**NOM**	**SAL**	**Verrou ligne**
0	Charlie	2000	X
1	Diana	2200	X
2	Eric	1600	

T_1 : UPDATE EMP SET SAL = SAL + 100
WHERE NE = 0

T_2 : UPDATE EMP SET SAL = SAL + 100
WHERE NE = 1 ← **EN ATTENTE !!**

Comment fluidifier les transactions ?

1. **Problème :**

   Poser juste des verrous X et S risque de bloquer trop de transactions

1. **Idée :**

   Créer des nouveaux types de verrous *au niveau table* mais annonçant des changements sur les lignes : RS et RX

ROW SHARE
ROW EXCLUSIVE

Verrous de type ROW-S et ROW-X

1. Un verrou de type RS ou RX pose un verrou au niveau table ET au niveau ligne (de type S ou X)
2. La décision **NON** est prise au niveau table → rapide
3. La décision **OUI** indique que la décision doit être validée au niveau ligne → pareil qu'avant

Verrou demandé sur table \ Verrou détenu sur table	RS	RX	S	X
RS	**oui**	**oui**	**oui**	non
RX	**oui**	**oui**	non	non
S	**oui**	non	**oui**	non
X	non	non	non	non

Verrou table RS / RX

1) verrous ligne

2) verrou table RX

EMP		Verrou : RX_1 RX_2	
NE	**NOM**	**SAL**	**Verrou ligne**
0	Charlie	2100	X
1	Diana	2200	X
2	Eric	1600	

T_1 : UPDATE EMP SET SAL = SAL + 100
WHERE NE = 0

T_2 : UPDATE EMP SET SAL = SAL + 100
WHERE NE = 1

ROW SHARE MODE (RS)

Explicite : LOCK TABLE *table* **IN ROW SHARE MODE;**

1. **Permet :** toutes les lectures de la table, et des modifications sur ce qu'on ne compte pas lire dans la table. (tout sauf X)

2. **Utilité :** on annonce qu'on va lire certaines données de la table. On ne bloque rien à part des personnes qui souhaiteraient également modifier ces mêmes données.

ROW EXCLUSIVE MODE (RX)

Explicite : LOCK TABLE *table* **IN ROW EXCLUSIVE MODE;**

Implicite :

INSERT INTO *table* **... ;**
UPDATE *table* **... ;**
DELETE FROM *table* **... ;**
SELECT ... FROM *table* **... FOR UPDATE ;**

- **Permet :** RS, RX
- **Utilité :** permet d'annoncer qu'on a apporté une modification à la table → empêche les verrous S et X sur les nuplets en question

SHARE MODE (S)

Explicite : LOCK TABLE *table* IN SHARE MODE;

1. **Permet** : RS, S (toutes les lectures)

2. **Utilité :** on veut que personne ne modifie la table sur laquelle on lit. Par contre on ne souhaite pas modifier la table.

   On ne bloque aucune lecture.

EXCLUSIVE MODE (X)

Explicite : LOCK TABLE *table* IN EXCLUSIVE MODE;

1. **Permet :** rien !

2. **Utilité :** bloque toutes les autres opérations sur la table.
   Utile par exemple pour une mise à jour globale de la table.

SHARE ROW EXCLUSIVE MODE (SRX)

Explicite : LOCK TABLE *table* IN SHARE ROW EXCLUSIVE MODE;

1. **Permet :** RS (uniquement une lecture sur ce qu'on n'a pas modifié)
2. **Utilité :** permet de faire S et RX en même temps : on bloque les modifications des autres sur toute la table *ET* on annonce une modification sur certaines lignes.

   Moins permissif que S

Conclusion sur le verrouillage hiérarchique

Pour obtenir	**Il faut avoir sur tous les ancêtres**
RS ou S	RS ou RX (avec S on l'a déjà)
RX, SRX ou X	RX ou SRX (avec X on l'a déjà)

Hiérarchie « BD »

base

↓

table

↓

tuple

Verrou demandé \ Verrou détenu	RS	RX	S	SRX	X
RS	**oui**	**oui**	**oui**	**oui**	non
RX	**oui**	**oui**	non	non	non
S	**oui**	non	**oui**	non	non
SRX	**oui**	non	non	non	non
X	non	non	non	non	non

CONCLUSION SUR LE VERROUILLAGE EN GENERAL

Approche pessimiste de la concurrence

1. Prévient les conflits
2. Assez coûteuse et assez complexe

Avantages du verrouillage

1. Abandon des seules transactions rendant l'exécution non sérialisable
2. Les performances sont bonnes : isolation paramétrable, taille variable du granule

Multi-versions

- En cas de mises-à-jour, on crée une nouvelle version
  - quand quelqu'un veut lire une entité, on lui fait lire une entité dans la version avant mise-à-jour
  - Avantage : les mises-à-jour ne ralentissent pas les transactions qui ne font que des lectures
  - on peut faire du 2PL ou de l'estampillage sur la version qui est mise-à-jour
- Régulièrement on bascule à une nouvelle version pour la lecture
- Intuition: Dans l'ordonnancement sériel équivalent, on regroupe toutes les transactions qui ne font que des lectures en gros paquets (qui correspondent à une version de lecture)
- Techniques sophistiquées pour éviter de dupliquer les données et
- pour jeter les versions devenues inutiles (GC)
- Exemple logiciel de gestion de versions comme SVN

Merci