

# Cours

## “Bases de données”

### Concurrence d'accès

3° année (MIS)

Antoine Cornuéjols

[www.lri.fr/~antoine](http://www.lri.fr/~antoine)  
[antoine.cornuejols@agroparistech.fr](mailto:antoine.cornuejols@agroparistech.fr)

## Contrôle de la concurrence d'accès

### 1. **Introduction**

- 1.1. Systèmes multi-utilisateurs
- 1.2. Pourquoi contrôler

### 2. **Concepts des transactions**

### 3. **Techniques de contrôle de la concurrence**

- 3.1. Verrouillage à deux phases
- 3.2. Contrôle par tri des estampilles
- 3.3. Autres problèmes

## Contrôle de la concurrence d'accès

### 1. **Introduction**

- 1.1. Systèmes multi-utilisateurs
- 1.2. Pourquoi contrôler

### 2. **Concepts des transactions**

### 3. **Techniques de contrôle de la concurrence**

- 3.1. Verrouillage à deux phases
- 3.2. Contrôle par tri des estampilles
- 3.3. Autres problèmes

## Contrôle de concurrence

### Introduction

#### GSBD Multi-utilisateurs

- Souvent des centaines, voire des milliers d'utilisateurs
  - E.g. : système de réservation de sièges dans les compagnies aériennes ; opérations bancaires ; ...
- Contraintes de temps réel

***Comment éviter les interactions négatives entre les différents utilisateurs et garantir la cohérence de la base de données ?***

# Contrôle de concurrence

## Introduction

### Transaction

- Programme formant une unité logique de traitement
- Souvent délimitée par des instructions de début et de fin de transaction

### Opérations élémentaires d'accès à la base de données :

#### • Lire\_element(x)

- lit un élément nommé  $x$  et le stocke dans une variable de programme.  
(On suppose que la variable de programme se nomme aussi  $x$ ).

#### • Ecrire\_element(x)

- écrit la valeur de la variable de programme  $x$  et le stocke dans l'élément de données  $x$ .

5

# Contrôle de concurrence

## Introduction

### Exemples :

```
T1
-----
lire_element(X);
X := X - N ;
ecrire_element(X);
lire_element(Y);
Y := Y + N ;
ecrire_element(Y);
```

Accède à {X, Y}

```
T2
-----
lire_element(X);
X := X + M ;
ecrire_element(X);
```

Accède à {X}

6

# Contrôle de concurrence

## Introduction : pourquoi contrôler ?

### Mises à jour perdues

T1	T2
lire_element(X); X := X - N;	
ecrire_element(X); lire_element(Y);	lire_element(X); X := X + M;
Y := Y + N; ecrire_element(Y);	ecrire_element(X);

L'élément  $X$  a une valeur incorrecte car sa mise à jour par T1 est "perdue" (écrasée).

7

# Contrôle de concurrence

## Introduction : pourquoi contrôler ?

### Mises à jour perdues

T1	T2
lire_element(X); X := 80 - 5;	
ecrire_element(75); lire_element(Y);	lire_element(X); X := 80 + 3;
Y := 20 + 5; ecrire_element(25);	ecrire_element(83);

L'élément  $X$  a une valeur incorrecte car sa mise à jour par T1 est "perdue" (écrasée).

X = 83 et Y = 25 au lieu de X = 78 et Y = 25

8

## Contrôle de concurrence

Introduction : pourquoi contrôler ?

*Mises à jour temporaires (ou "lectures sales")*

T1	T2
<pre>lire_element(X); X := X-N; ecrire_element(X);</pre>	<pre>lire_element(X); X := X + M; ecrire_element(X);</pre>
<pre>lire_element(Y);</pre>	

La transaction T1 échoue et doit rétablir l'ancienne valeur de X ;  
au même moment, T2 a lu la valeur "temporaire"

9

## Anomalies transactionnelles

Lecture impropre

L'utilisateur 1 ajoute 10 places et annule sa transaction, l'utilisateur 2 veut 7 places si elles sont disponibles...

```

T1 BEGIN TRANSACTION 1
T3 UPDATE T_VOL
SET VOL_PLACES_LIBRES = VOL_PLACES_LIBRES + 10
WHERE VOL_ID = 2
T5 ROLLBACK TRANSACTION 1

T2 BEGIN TRANSACTION 2
T4 IF (SELECT VOL_PLACES_LIBRES
FROM T_VOL WHERE VOL_ID = 2) >= 7
then
T6 BEGIN
UPDATE T_VOL
SET VOL_PLACES_LIBRES = VOL_PLACES_LIBRES - 7
WHERE VOL_ID = 2
T7 INSERT INTO T_CLIENT_VOL VALUES (77, 2, 5)
T8 COMMIT TRANSACTION
end
T9 ELSE
ROLLBACK TRANSACTION 2
    
```

Temps	T_VOL :			T_CLIENT_VOL :			TRANSACTION
	VOL_ID	VOL_REFERENCE	VOL_PLACES_LIBRES	CLI_ID	VOL_ID	VOL_PLACES_PRISES	
T1	2	AF 812	6				DEBUT TRANSACTION POUR UTILISATEUR 1
T2	2	AF 812	6				DEBUT TRANSACTION POUR UTILISATEUR 2
T3	2	AF 812	16				
T4	2	AF 812	16				
T5	2	AF 812	6				FIN TRANSACTION POUR UTILISATEUR 1 AVEC ROLLBACK
T6	2	AF 812	-1				
T7	2	AF 812	-1	77	2	5	
T8	2	AF 812	-1	77	2	5	FIN TRANSACTION POUR UTILISATEUR 2 AVEC COMMIT

## Contrôle de concurrence

Introduction : pourquoi contrôler ?

*Résumés incorrects*

T1	T3
<pre>lire_element(X); X := X-N; ecrire_element(X);</pre>	<pre>somme := 0; lire_element(A); somme := somme + A; . . . lire_element(X); somme := somme + X; lire_element(Y); somme := somme + Y;</pre>
<pre>lire_element(Y); Y := Y + N; ecrire_element(Y);</pre>	

T3 lit X après que N ait été soustrait et lit Y avant que N ne soit ajouté.  
Il en résulte un résumé incorrect (différence de N).

## Anomalies transactionnelles

Lecture non répétable

Cas où notre opérateur désire toutes les places d'un vol si il y en a plus de 4...

```

T1 BEGIN TRANSACTION 2
T2 IF (SELECT VOL_PLACES_LIBRES
FROM T_VOL WHERE VOL_ID = 2) >= 4
then
T6 BEGIN
UPDATE T_VOL
SET VOL_PLACES_LIBRES = VOL_PLACES_LIBRES -
(SELECT VOL_PLACES_LIBRES
FROM T_VOL WHERE VOL_ID = 2)
WHERE VOL_ID = 2
T7 INSERT INTO T_CLIENT_VOL VALUES (77, 2,
(SELECT VOL_PLACES_LIBRES
FROM T_VOL WHERE VOL_ID = 2))
T8 COMMIT TRANSACTION
end
T9 ELSE
ROLLBACK TRANSACTION 2
    
```

Temps	T_VOL :			T_CLIENT_VOL :			TRANSACTION
	VOL_ID	VOL_REFERENCE	VOL_PLACES_LIBRES	CLI_ID	VOL_ID	VOL_PLACES_PRISES	
T1	2	AF 812	6				DEBUT TRANSACTION POUR UTILISATEUR 1
T2	2	AF 812	6				DEBUT TRANSACTION POUR UTILISATEUR 2
T3	2	AF 812	16				
T4	2	AF 812	2				
T5	2	AF 812	2				FIN TRANSACTION POUR UTILISATEUR 1 AVEC COMMIT
T6	2	AF 812	0				
T7	2	AF 812	0	77	2	2	
T8	2	AF 812	0	77	2	2	FIN TRANSACTION POUR UTILISATEUR 2 AVEC COMMIT

## Anomalies transactionnelles

### Lecture fantôme

Notre utilisateur 2 désire n'importe quel vol pas cher pour emmener son équipe de foot (soit 11 personnes) à une destination quelconque. Résultat : 11 places volatilisées du vol AF111 (histoire hélas véridique...!)

```

T1 BEGIN TRANSACTION 2
T2 if EXISTS (SELECT *
            FROM T_VOL
            WHERE VOL_PLACE_LIBRE >= 11) then
T6 begin
    UPDATE T_VOL
    SET VOL_PLACES_LIBRES = VOL_PLACES_LIBRES - 11
    WHERE VOL_PLACES_LIBRES >= 11
T7 INSERT INTO T_CLIENT_VOL VALUES (77, 4, 11)
T8 COMMIT TRANSACTION
    end
T9 else
    ROLLBACK TRANSACTION 2
    
```

Temps	T_VOL :			T_CLIENT_VOL :			TRANSACTION
	VOL_ID	VOL_REFERENCE	VOL_PLACES_LIBRES	CLI_ID	VOL_ID	VOL_PLACES_PRISES	
T1	4	AF 325	258				DEBUT TRANSACTION POUR UTILISATEUR 1
T2	4	AF 325	258				
T3	4	AF 325	258				DEBUT TRANSACTION POUR UTILISATEUR 2
T4	4	AF 325	258				
T5	4	AF 325	258	5	AF 111	125	FIN TRANSACTION POUR UTILISATEUR 1 AVEC COMMIT
	5	AF 111	125				
T6	4	AF 325	247				
	5	AF 111	114				
T7	4	AF 325	247	77	4	11	
	5	AF 111	114				
T8	4	AF 325	247	77	2	11	FIN TRANSACTION POUR UTILISATEUR 2 AVEC COMMIT
	5	AF 111	114				

## Anomalies transactionnelles

### Lecture fantôme

## Contrôle de concurrence

### Introduction : Isolation des transactions

#### Phénomènes indésirables :

- 🕒 **Lecture sale** : Une transaction lit des données écrites par une transaction concurrente mais non validée
- 🕒 **Lecture non reproductible** : Une transaction lit de nouveau des données qu'il a lues précédemment et trouve que les données ont été modifiées par une autre transaction (qui a validé depuis la lecture initiale)
- 🕒 **Lecture fantôme** : Une transaction ré-exécute une requête renvoyant un ensemble de lignes satisfaisant une condition de recherche et trouve que l'ensemble des lignes satisfaisant la condition a changé à cause d'une transaction récemment validée.

## Contrôle de la concurrence d'accès

### 1. Introduction

- 1.1. Systèmes multi-utilisateurs
- 1.2. Pourquoi contrôler

### 2. Concepts des transactions

### 3. Techniques de contrôle de la concurrence

- 3.1. Verrouillage à deux phases
- 3.2. Contrôle par tri des estampilles
- 3.3. Autres problèmes

## Contrôle de concurrence

### Introduction : Importance de la reprise

#### *Le SGBD doit vérifier que :*

- toutes les opérations de la transaction ont bien réussi et que leurs effets ont bien été enregistrés dans la base de données

ou bien que :

- la transaction n'a aucun effet sur la base de données ou sur toute autre transaction

17

## Contrôle de concurrence

### Introduction : Importance de la reprise

#### *Types de pannes :*

- *Panne informatique (crash système)* : affectent surtout la mémoire centrale
- *Erreur de la transaction ou erreur système* : (e.g. erreur d'opération (débordement, division par zéro))
- *Erreur locale ou conditions d'exception détectées par la transaction* : ne doit pas être considérée comme une panne mais traitée au niveau de la transaction
- *Application du contrôle de la concurrence* : E.g. si situation d'interblocage
- *Panne de disque*
- *Problèmes physiques ou catastrophes*

18

## Contrôle de concurrence

### Concept de transaction

#### **Transaction**

- Unité atomique de travail
  - soit totalement terminée ; soit pas du tout.

#### *Opérations surveillées :*

- DEBUT\_TRANSACTION
- LIRE ou ECRIRE
- FIN\_TRANSACTION
- COMMIT signale que la transaction s'est bien terminée.
- ROLL-BACK signale que la transaction ne s'est pas bien terminée. Toutes les modifications effectuées doivent être annulées.

19

## Contrôle de concurrence

### Concept de transaction

Les opérations de transaction sont écrites dans un *journal* archivé sur disque dur et sur sauvegarde.

- [début\_transaction, T] : indique que la transaction a démarré
- [ecrire\_element, T, x, ancienne\_valeur, nouvelle\_valeur]
- [lire\_element, T, x]
- [valider, T] : indique que la transaction s'est terminée sans problème et que ses opérations peuvent être validées (enregistrées de façon permanente dans la base de données)
- [annuler, T] : indique que T a été annulée.

20

## Contrôle de concurrence

### Concept de transaction

#### Les propriétés ACID

- ☛ **Atomicité.** Une transaction est une unité atomique de traitement.
- ☛ **Cohérence.** Une transaction préserve la cohérence si son exécution complète fait passer la base de données d'un état cohérent à un autre.
- ☛ **Isolation.** Les résultats d'une transaction doivent être invisibles pour les autres transactions tant qu'elle n'est pas validée. Autrement dit, les exécutions ne doivent pas interférer les unes avec les autres.
- ☛ **Durabilité.** Les changements appliqués à la BD par une transaction validée doivent persister dans celle-ci. Ces changements ne doivent pas être perdus, même à la suite d'une défaillance.

21

## Contrôle de concurrence

### Concept de transaction

#### Sérialisabilité

Les accès simultanés aux mêmes objets d'une BD doivent être sérialisés pour que ses utilisateurs puissent travailler indépendamment les uns des autres.

Un ensemble de transactions concurrentes est correctement synchronisé si leur exécution séquentielle génère un état de la BD identique à celui qui serait obtenu si elles étaient exécutées indépendamment (par un seul utilisateur).

22

## Contrôle de concurrence

### Concept de transaction

#### Équivalence d'exécutions

Deux exécutions sont équivalentes si :

- ☛ Ont les mêmes transactions et les mêmes opérations
- ☛ Produisent les mêmes effets sur la base
- ☛ Produisent les mêmes effets dans les transactions

$T_1 = r_1[x] \ x=x*5 \ w_1[x] \ C_1$

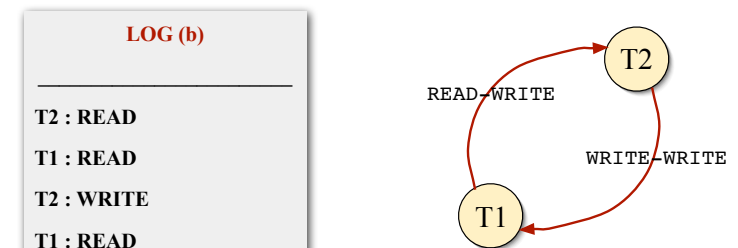
$T_2 = r_2[x] \ x=x+10 \ w_2[x] \ C_2$

$T_1 T_2$  non équivalent à  $T_2 T_1$

23

## Contrôle de concurrence

### Sérialisabilité



Grappe de précédence  
pour la donnée  $b$

24

## Contrôle de concurrence

### Sérialisabilité

#### Critère de sérialisabilité

Un ensemble de transactions est sérialisable si le graphe de précédence correspondant ne contient *aucun cycle*.

25

## Traitement et optimisation des requêtes

### Exercices

Détecter les conflits et construire les graphes de sérialisation pour les exécutions suivantes. Indiquer les exécutions sérialisables et vérifier s'il y a des exécutions équivalentes.

H1: w2[x] w3[z] w2[y] c2 r1[x] w1[z] c1 r3[y] c3

H2: r1[x] w2[y] r3[y] w3[z] c3 w1[z] c1 w2[x] c2

H3: w3[z] w1[z] w2[y] w2[x] c2 r3[y] c3 r1[x] c1

## Contrôle de concurrence

### Concept de transaction

#### Prise en charge par SQL

- Pas d'instruction Debut\_Transaction
- Chaque instruction doit avoir une instruction finale explicite : COMMIT ou ROLLBACK.
- Chaque transaction possède certaines caractéristiques spécifiées par une instruction SET TRANSACTION.
  - *Mode d'accès, taille, taille de la zone de diagnostic, niveau d'isolation.*

27

## Contrôle de concurrence

### Concept de transaction

#### Prise en charge par SQL

Niveau d'isolation	Type de violation		
	Lecture sale	Lecture non reproductible	Lecture fantôme
READ UNCOMMITTED	Oui	Oui	Oui
READ COMMITTED	Non	Oui	Oui
REPEATABLE READ	Non	Non	Oui
SERIALIZABLE	Non	Non	Non

28

## Contrôle de concurrence

### Concept de transaction

#### Exemple de transaction SQL

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTIC SIZE 5
    ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPLOYE (PRENOM, NOM, NOSS, NoSce, SALAIRE)
    VALUES ('Robert', 'Forgeron', '1550222111999', 2, 35000);
EXEC SQL UPDATE EMPLOYE
    SET SALARY = SALARY * 1.1 WHERE NOSERVICE = 2;
EXEC SQL COMMIT;
GOTO LA_FIN;
UNDO: EXEC SQL ROLLBACK;
LA_FIN: ...;
```

29

## Contrôle de la concurrence d'accès

### 1. Introduction

- 1.1. Systèmes multi-utilisateurs
- 1.2. Pourquoi contrôler

### 2. Concepts des transactions

### 3. Techniques de contrôle de la concurrence

- 3.1. Verrouillage à deux phases
- 3.2. Contrôle par tri des estampilles
- 3.3. Autres problèmes

## Contrôle de concurrence

### Méthodes de contrôle

Deux types de méthodes :

#### 🔑 Contrôle continu :

- on vérifie **au fur et à mesure** de l'exécution des opérations que le critère de sérialisabilité est bien respecté. Ces méthodes sont dites *pessimistes* : elles reposent sur l'idée que les conflits sont fréquents et qu'il faut les traiter le plus tôt possible.

#### 🔑 Contrôle par certification :

- cette fois, on se contente de vérifier la sérialisabilité **quand la transaction s'achève**. Il s'agit d'une approche dite *optimiste* : on considère que les conflits sont rares et que l'on peut accepter de ré-exécuter les quelques transactions qui posent problèmes.

31

## Contrôle de concurrence

### Méthodes de contrôle

Le contrôle continu est la méthode la plus employée.

On utilise alors des *méthodes de verrou*

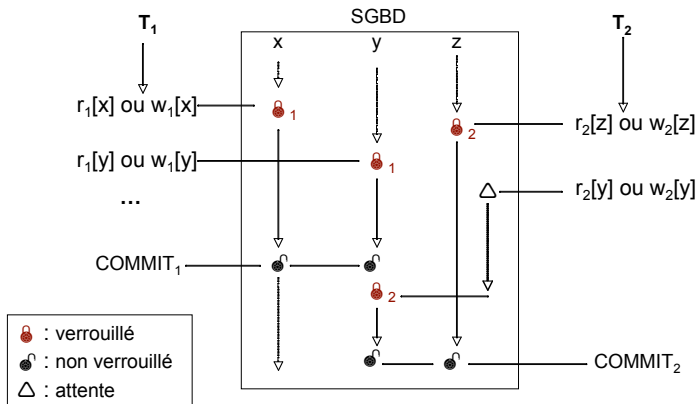
Le principe est simple :

- 🔑 on **bloque l'accès** à une donnée dès qu'elle est lue ou écrite par une transaction ("*pose de verrou*")
- 🔑 on **libère cet accès** quand la transaction se termine par commit ou rollback ("*libération du verrou*").

32

## Contrôle de concurrence

Méthodes de contrôle : méthode des verrous



33

## Contrôle de concurrence

Méthodes de contrôle : méthode de verrouillage

### Granularité

Le verrouillage peut intervenir à **différents niveaux** de la base:

- Verrouillage de la **base de données**
- Verrouillage des **tables** concernées
- Verrouillage au niveau de la **page**
- Verrouillage des **lignes** (mais une surcharge qui peut être supérieure au gain de performance)
- ... Verrouillage individuel des **données**

34

## Contrôle de concurrence

Méthodes de contrôle : méthode de verrouillage

Il existe essentiellement deux degrés de restriction.

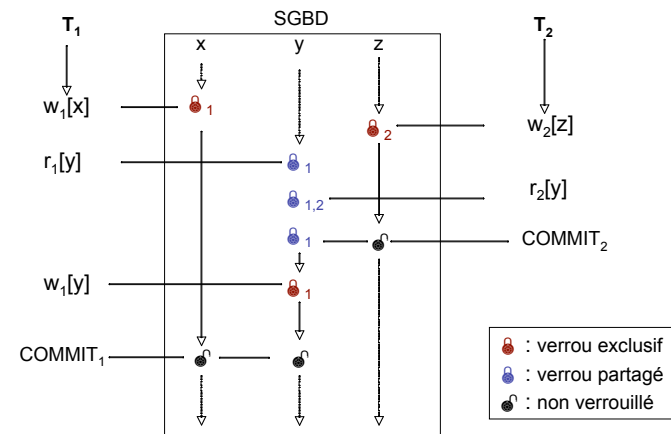
- Le **verrou partagé** ("shared lock") est typiquement utilisé pour permettre à plusieurs transactions concurrentes de **lire** la même ressource.
- Le **verrou exclusif** ("exclusive lock") réserve la ressource en **écriture** à la transaction qui a posé le verrou.

Ces verrous sont posés de manière automatique par le SGBD en fonction des opérations effectuées par les transactions ou les utilisateurs. Mais il est également possible de demander explicitement le verrouillage de certaines ressources.

35

## Contrôle de concurrence

Méthodes de contrôle : méthode des verrous



36

## Contrôle de concurrence

### Méthodes de contrôle : méthode de verrouillage

L'ordonnanceur ("scheduler") a la charge de transformer les requêtes READ et WRITE qu'il reçoit du préprocesseur en requêtes SREAD ou XREAD et XWRITE, puis d'ajouter des requêtes SRELEASE ou XRELEASE selon les protocoles suivants:

#### Verrouillage en mode X (exclusif)

- Une transaction  $T$  qui veut accéder à une granule de données  $D$  pour **mise à jour** doit d'abord faire un XREAD  $D$  qui a pour effet:
  - si  $D$  est non verrouillé ni au niveau  $X$  ni même au niveau  $S$ , alors de le verrouiller au niveau  $X$  sinon de suspendre l'exécution de  $T$  jusqu'au déverrouillage de  $D$  par la transaction concurrente qui l'avait verrouillé à ce niveau.

37

## Contrôle de concurrence

### Méthodes de contrôle : méthode de verrouillage

#### Verrouillage en mode S (partagé)

- Une transaction  $T$  qui veut accéder à une granule de données  $D$  pour **lecture reproductible** doit d'abord faire un SREAD  $D$  qui a pour effet:
  - si  $D$  est verrouillé au niveau  $X$  alors de suspendre l'exécution de  $T$  jusqu'au déverrouillage  $\_XRELEASE\_$  de  $D$  par la transaction concurrente qui l'avait verrouillé, sinon  $T$  est ajoutée à la liste des transactions qui ont verrouillé  $D$  au niveau  $S$ .  
 $T$  ne peut elle-même faire une mise à jour de  $D$  que par un XWRITE  $D$  qui tente de verrouiller au niveau  $X$  ce qui peut la suspendre jusqu'après le dernier SRELEASE de  $D$  des transactions concurrentes.

38

## Contrôle de concurrence

### Méthodes de contrôle : méthode de verrouillage

#### Règles associées aux verrous

- Un verrou peut toujours être posé **sur une donnée non verrouillée**, la donnée est alors verrouillée
- Plusieurs verrous en lecture peuvent être posés **sur une donnée déjà verrouillée en lecture**
- **Si une donnée est verrouillée en écriture**, aucun autre verrou ne peut être posé tant que ce verrou n'a pas été supprimé.

Deux verrous sont dits *compatibles* si deux transactions qui accèdent à la même donnée peuvent obtenir les verrous sur cette donnée au même moment.

39

## Contrôle de concurrence

### Méthodes de contrôle : méthode de verrouillage

#### Compatibilité entre verrous

	$Lock(T_i, x, Lecture)$	$Lock(T_i, x, Ecriture)$
$Lock(T_j, x, Lecture)$	compatible	non compatible
$Lock(T_j, x, Ecriture)$	non compatible	non compatible

40

## Contrôle de concurrence

### Le verrouillage à deux phases

Le respect des deux protocoles précédents n'interdit pas des verrouillages et déverrouillages au fur et à mesure de la progression des transactions.

Cela laisse la porte ouverte à de nombreux cas d'interblocage de toutes les transactions - appelés *verrous mortels* - ou de blocage éternel de quelques unes - appelés *famines*.

On démontre que si l'exécution de chaque transaction en **deux phases seulement** est imposée, une **première** pour tous les *verrouillages* \_SREAD, XREAD ou XWRITE\_ une **seconde** pour tous les *déverrouillages* \_SRELEASE, XRELEASE\_ sans qu'il soit possible de verrouiller à nouveau, il existe alors au moins une exécution globale sérialisable. De plus certains cas de verrou mortel ou de famine sont aussi implicitement résolus.

41

## Contrôle de concurrence

### Le verrouillage à deux phases

Le plus répandu.

$wl_i[x]$  : verrou en écriture (*write lock*)

$rl_i[x]$  : verrou en lecture (*read lock*)

#### Conflit

Deux verrous  $pl_i[x]$  et  $ql_j[y]$  sont en **conflit** si  $x = y$  et  $pl$  ou  $ql$  est un verrou en écriture.

#### Stratégie de l'ordonnanceur

- **Règle 1** : L'ordonnanceur reçoit  $p_i[x]$  et consulte le verrou déjà posé sur  $x$ ,  $ql_i[x]$ , s'il existe.
  - si  $pl_i[x]$  est en conflit avec  $ql_i[x]$ ,  $p_i[x]$  est retardée et la transaction  $T_i$  est mise en attente.
  - sinon,  $T_i$  obtient le verrou  $pl_i[x]$  et l'opération  $p_i[x]$  est exécutée.
- **Règle 2** : Un verrou pour  $p_i[x]$  n'est jamais relâché avant la confirmation de l'exécution par un autre module, le gestionnaire de données.
- **Règle 3** : Dès que  $T_i$  relâche un verrou, elle ne peut plus en obtenir d'autre.

42

## Contrôle de concurrence

### Le verrouillage à deux phases

#### “Verrouillage à deux phases”

D'abord **accumulation** de verrous pour une transaction  $T_i$ , puis **libération** des verrous.

Théorème :

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

43

## Traitement et optimisation des requêtes

### Exercices

Un Ordonnanceur avec verrouillage à deux phases reçoit la séquence d'opérations ci-dessous :

H1:  $r1[x]$   $r2[y]$   $w3[x]$   $w1[y]$   $w1[x]$   $w2[y]$   $c2$   $r3[y]$   $r1[y]$   $c1$   $w3[y]$   $c3$

Indiquez l'ordre d'exécution établi par l'Ordonnanceur, en considérant qu'une opération bloquée en attente d'un verrou est exécutée en priorité dès que le verrou devient disponible. On suppose que les verrous d'une transaction sont relâchés au moment du COMMIT.

## Contrôle de concurrence

### Le verrouillage à deux phases

#### Interblocage

Soient les deux transactions :

- $T1 : r1[x] \rightarrow w1[y] \rightarrow c1$
- $T2 : r2[x] \rightarrow w2[y] \rightarrow c2$

T1 et T2 s'exécutent en concurrence dans le cadre d'un verrouillage à deux phases :

$r11[x] \ r1[x] \ r12[y] \ r2[y] \ w11[y] \ T1 \text{ en attente} \ w12[x] \ T2 \text{ en attente}$

T1 et T2 sont en attente l'une de l'autre : **interblocage**

Évitable avec détection de cycle dans le graphe de dépendance.

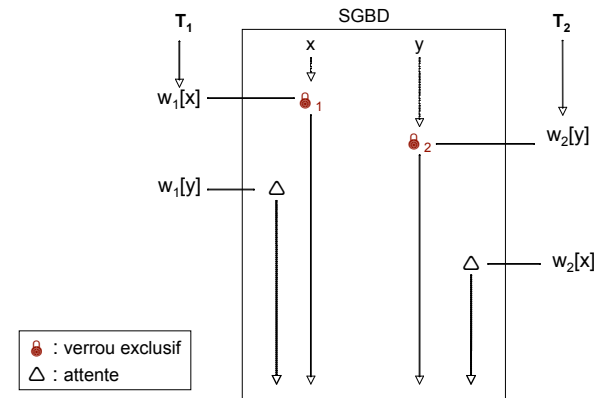
On élimine alors une transaction, appelée *victime*.

45

## Contrôle de concurrence

### Le verrouillage à deux phases

#### Verrou mortel



46

## Contrôle de concurrence

### Le verrouillage à deux phases

#### Verrou mortel

##### Prévention :

- Estampillage des transactions : ordonner strictement les transactions (horodate de lancement)
- WAIT-DIE : annuler les transactions qui demandent des ressources tenues par des transactions plus anciennes
- WOUND-WAIT : annuler les transactions qui tiennent des ressources demandées par des transactions plus anciennes

##### Détection :

- Graphes d'attente : annuler les transactions (et libérer les verrous correspondants) qui provoquent des cycles
- Mécanismes de timeout

47

## Contrôle de concurrence

### Les verrous sous Oracle

#### Types de verrous

##### Verrous implicites :

- SELECT : pas de pose de verrou
- ALTER : pose d'un verrou
- INSERT, UPDATE, DELETE : verrouillage **row exclusive** de la table, puis verrouille les nuplets impliqués
- SELECT ... FOR UPDATE fait de même mais pose un verrou intentionnel **row share** sur la table.  
Si une telle exécution tente de poser un verrou sur un nuplet déjà verrouillé par une autre transaction, elle est bloquée jusqu'à la terminaison de cette autre transaction. Lors du déblocage l'ensemble des nuplets sélectionnés est réévalué.

48

## Contrôle de concurrence

### Les verrous sous Oracle

#### 🔑 **Verrous explicites :**

- 👤 Il est parfois nécessaire de verrouiller explicitement les données avec la commande LOCK TABLE.

```
lock table <table> in { exclusive | share } mode [nowait] ;
```

#### 👤 4 catégories importantes

- **Verrous exclusifs** : INSERT, UPDATE, DELETE
  - Les autres transactions ne peuvent que lire la table (mise à jour et pose de verrous bloquées)
- **Verrous partagés (share)**
  - Bloque, pour les autres transactions, la pose du verrou intentionnel row exclusive, et la pose de verrou exclusive
- **Verrous partagés en mise à jour (share update)**
- **Verrous d'attente (NOWAIT)**
  - Plutôt que de bloquer une table déjà verrouillée, Oracle abandonne l'instruction en signalant l'erreur -54 : la transaction peut alors faire autre chose avant de retenter le verrouillage.

49

## Contrôle de concurrence

### Les verrous sous Oracle

#### **Mise d'un verrou**

##### Commande LOCK TABLE

##### Options :

- 🔑 **Table**
- 🔑 **Type de verrou** : share, share\_update, exclusif
- 🔑 **Mode** : nowait pour dire de mettre le verrou toute de suite ou d'attendre

Le verrouillage se fait de la manière suivante **en mode nowait** si on veut mettre un verrou sans attendre un commit (par exemple si un Tony veut mettre un verrou sans attendre que bob finisse ces manipulations en faisant un commit.) :

```
lock table Magasin in exclusive mode [nowait] ;
```

Sinon : 

```
lock table Magasin in exclusive mode ;
```

50

## Contrôle de concurrence

### Les verrous sous Oracle

#### **Retrait d'un verrou**

Commande COMMIT ou lors d'un Rollback

51

## Contrôle de concurrence

### Exercices

52

## Contrôle de concurrence

### Contrôle de concurrence : méthodes optimistes

53

## Contrôle de concurrence

### Contrôle de concurrence : méthodes par estampillage

Le mécanisme de gestion des concurrences par verrouillage à deux phases détecte *a posteriori* des attentes entre transactions. Des mécanismes de prévention basés sur un ordonnancement *a priori* des actions composantes des transactions peuvent être aussi envisagés.

L'**estampillage** est un mécanisme du type prévention.

Les transactions  $T$  reçoivent de l'ordonnanceur à leur début - Begin Of Transaction - une estampille qui est un numéro d'ordre de présentation en entrée de l'ordonnanceur. Chaque granule  $G$  de données note l'estampille de la dernière transaction qui y a accédé.

Un granule n'accepte un nouvel accès que s'il est demandé par une transaction "plus jeune", c'est-à-dire de numéro d'estampille plus grand.

54

## Contrôle de concurrence

### Contrôle de concurrence : méthodes par estampillage

L'**estampillage** est un mécanisme du type prévention.

Les transactions  $T$  reçoivent de l'ordonnanceur à leur début - Begin Of Transaction - une estampille qui est un numéro d'ordre de présentation en entrée de l'ordonnanceur. Chaque granule  $G$  de données note l'estampille de la dernière transaction qui y a accédé.

Un granule n'accepte un nouvel accès que s'il est demandé par une transaction "plus jeune", c'est-à-dire de numéro d'estampille plus grand.

Lire(T,G,X)	Ecrire(T,G,X)
<u>Si</u> Estampille(T) >= Estampille(G) <u>alors</u> X <- lecture(G) Estampille(G) := Estampille(T) <u>Sinon</u> rollback(T)	<u>Si</u> Estampille(T) >= Estampille(G) <u>alors</u> écriture(G) <- X Estampille(G) := Estampille(T) <u>Sinon</u> rollback(T)

La transaction rejetée est reprise par l'ordonnanceur après avoir reçu de lui une nouvelle estampille.

55

## Contrôle de concurrence

### Contrôle de concurrence : méthodes par estampillage

Les SGBD commerciales offrent des fonctions avancées de verrouillage

- **Verrouillage explicite** : un programme peut explicitement verrouiller une partie de la base de données
- **Niveaux d'isolation** : le SGBD peut libérer les verrous avant COMMIT
- **Paramètres de verrouillage** : taille du verrou, nombre de verrous, etc. ; afin d'optimiser les performances du verrouillage

56

## Résistance aux pannes

Plusieurs sources de pannes dans un SGBD

- Panne d'une action
- Panne d'une transaction
- Panne du système
- Panne de mémoire secondaire

But : minimiser le travail perdu tout en assurant un retour à des données cohérentes

- Points de sauvegarde (*savepoints*)
- Journal des transactions
- Validation, annulation et reprise de transaction