

CONTROLE

DE LA

CONCURRENCE

Exécution concurrente de programmes

Programme : Retrait Distributeur

Paramètre : numéro de compte (C)
montant retrait (R)

début **Lire** le solde du compte (Solde)
 si Solde \geq R **alors**
 accepter le retrait; délivrer le montant demandé
fin **Ecrire** le nouveau solde Solde-R

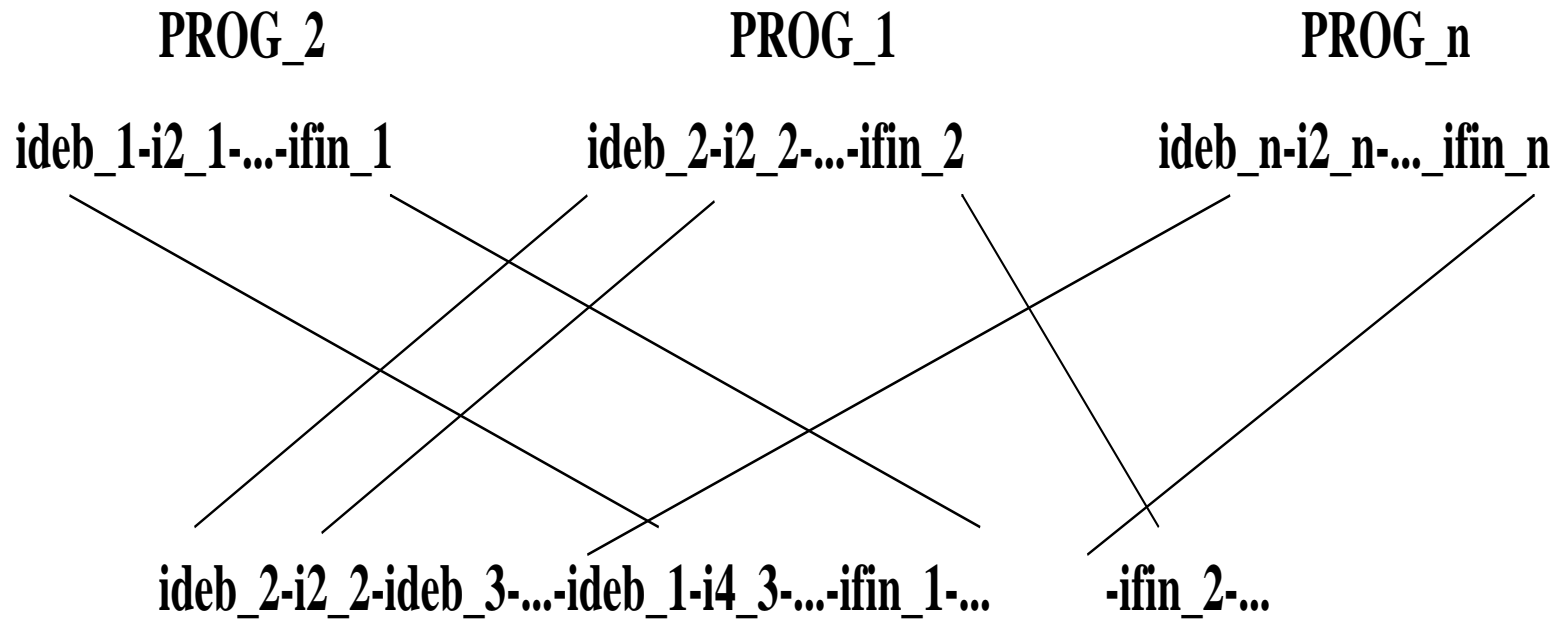
- **exécution** d'un programme d'application
- **Ce qui compte : Lecture et Ecriture**

Les instructions comme les tests, les calculs, ... n'ont pas d'importance pour le contrôle de la concurrence.

Exécution du programme : **R**(Solde) **W**(Solde)

R : lecture (Read) , **W** : écriture (Write) dans la base de données.

Exécution concurrente de programmes



Plusieurs exécutions du MÊME programme sont possibles :

R(Solde) **R**(Solde) **W**(Solde) **W**(Solde)

Exécution Concurrente de transactions : pourquoi ?

- entrées/sorties parallélisables avec les calculs (CPU)
→ réduire les temps de non utilisation du disque / processeur

-
- faire qu'un programme court puisse être exécuté pendant un programme long

→ contrôle du temps de réponse

Transactions et propriétés ACID

- une transaction = suite de lectures et écritures correspondant à une exécution d'un programme d'application

- **atomicité** : soit toutes les actions sont exécutées soit aucune n'est exécutée !

L'utilisateur ne doit pas avoir à gérer les cas d'interruption d'une transaction (à cause d'une erreur, d'une panne, etc ...)

- **cohérence** : exécuter seule sur une base de données satisfaisant les CIs, une transaction produit une base de données cohérente

Le programmeur d'application est essentiellement le garant de cette propriété.

Les propriétés ACID d'une transaction

■ **isolation** : Le résultat d'une transaction doit être comparable à celui produit dans un environnement mono-utilisateur.

L'utilisateur ne doit pas avoir à prendre en compte l'interaction avec d'autres programmes pour écrire ses procédures de traitement.

■ **durabilité** : Une fois terminée, les effets d'une transaction doivent persister malgré les pannes ou tout autre problème du sgbd.

L'utilisateur ne doit pas avoir à prendre en compte ce qui peut se passer après la fin (normale) d'un programme.

Transaction : exécution incomplète

- erreur dans la transaction
- anomalie de fonctionnement de l'application détectée par le sgbd
- panne logicielle
- panne matérielle

2 actions en +

- **validation** (commit): action au delà de laquelle une transaction est considérée comme terminée
- **annulation** (abort) : action qui termine également une transaction mais en annulant tous les effets antérieurs

Analyse STATIQUE du problème

Etant donnée une exécution concurrente
celle-ci est-elle “correcte” ?

Quelques **anomalies** bien connues :

- Lecture non validée conflit Write-Read
- Lecture non reproductible conflit Read-Write
- Perte d’opération conflit Write-Write

Lecture non validée (dirty read)

T_1	R	W					R	W	com
	(A)	(A)					(B)	(B)	
T_2			R	W	R	W	com		
			(A)	(A)	(B)	(B)			

La transaction T_1 peut écrire dans A une valeur rendant la base de donnée incohérente.

Lecture non validée (dirty read)

T_1	R	W	abort	
	(A)	(A)		
T_2		R	W	...
		(A)	(A)	...

La transaction T_2 a lue une valeur de A modifiée par T_1 , valeur qui n'a jamais existé !

Lectures non reproductibles (non repeatable read)

T_1	R			R	...	com
	(A)			(A)	...	
T_2		R	W			com
		(A)	(A)			

La transaction T_1 peut se trouver lire deux valeurs distinctes de A sans avoir modifié A (impossible en isolation).

Perte d'opération

T_1	R			W	com
	(A)			(A)	
T_2		R	W	com	
		(A)	(A)		

La transaction T_1 écrase l'écriture de la transaction T_2 !

Annulation impossible (unrecoverable schedule)

T_1	R	W						abort
	(A)	(A)						
T_2			R	W	R	W	com	
			(A)	(A)	(B)	(B)		

L'annulation de la transaction T_1 doit entraîner l'annulation de la transaction T_2 pour éviter les problèmes de valeur fantôme. Ceci est impossible car la transaction T_2 est validée !

Exécution concurrente correcte : sérialisabilité

- on ne s'occupe pas des annulations (pour le moment)

Observer les conflits !

- Actions conflictuelles :

T_1	T_2
R(A)	W(A)
W(A)	R(A)
W(A)	W(A)

S est une exécution concurrente des transactions T_1, T_2, \dots, T_n

S est **conflictuellement sérialisable** si les actions conflictuelles de S sont exécutées dans le même ordre que les actions conflictuelles d'une exécution en série de T_1, T_2, \dots, T_n .

Un exemple d'exécution concurrente sérialisable

T_1	R (A)	W (A)			R (B)	W (B)		
T_2			R (A)	W (A)			R (B)	W (B)

Une exécution en série

T_1	R (A)	W (A)	R (B)	W (B)				
T_2					R (A)	W (A)	R (B)	W (B)

Sérialisabilité : une solution pratique

Graphe des conflits :

- un noeud pour chaque transaction
 - un arc de T_i vers T_j si il existe une action a de T_i qui précède une action b de T_j telles que a et b soient conflictuelles.
-

S est sérialisable

ssi

le graphe des conflits de S est **acyclique**

Analyse dynamique du problème

Pourquoi ne pas utiliser le test de sérialisabilité ?

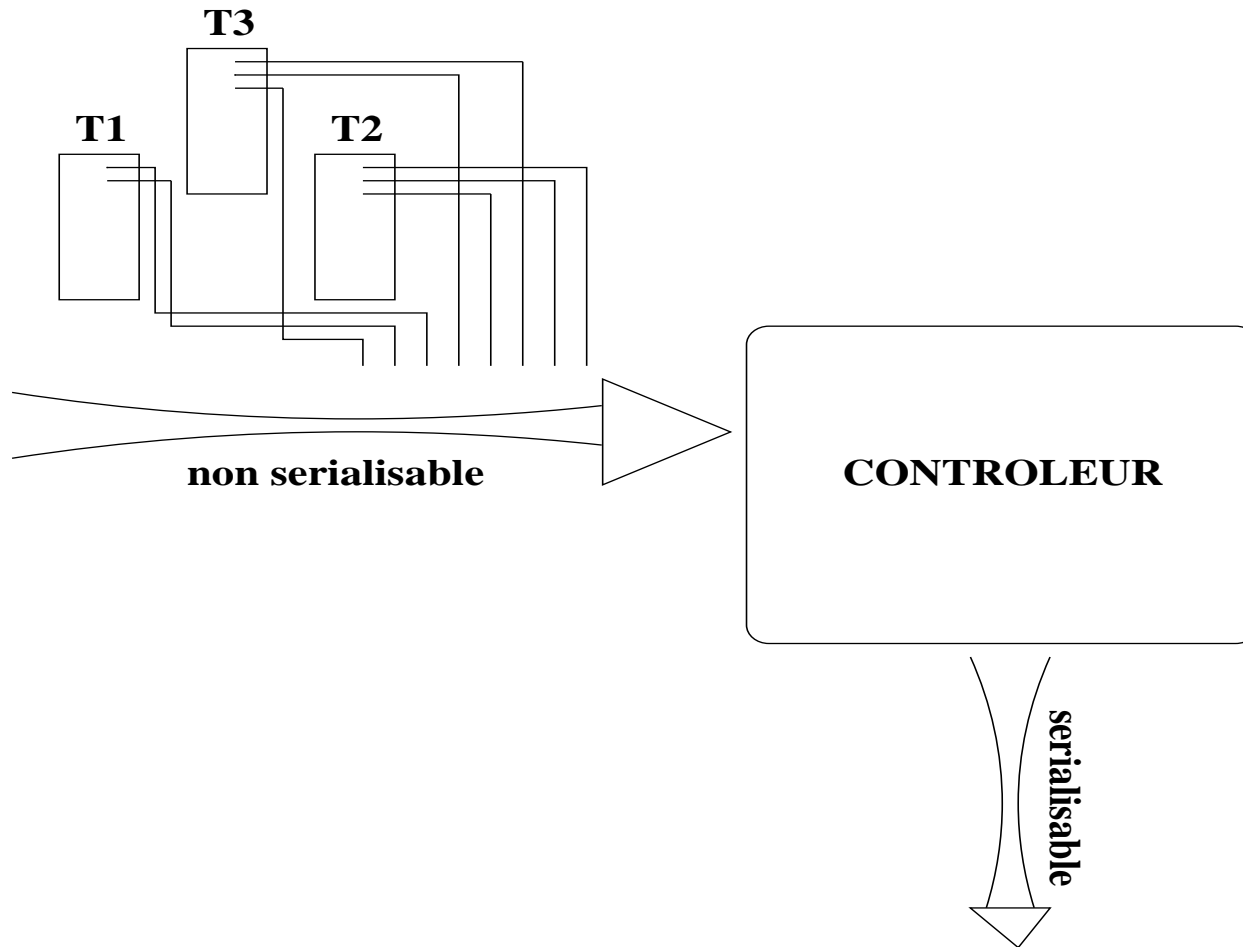
Qu'est-ce que le contrôle dynamique ?

A quoi sert l'analyse statique ?

Méthodes :

- **Verrouillage en 2 phases**
- **Estampillage**

Contrôle de la concurrence



Aglo “contrôleur”

```
declare der-etape , att-etape : etape ; file-etape : file of etape ;  
begin file-etape :=  $\emptyset$  ;  
    repeat on arrival der-etape do  
        mise-a-jour-etat ;  
        control(der-etape) ;  
        for each att-etape := tete(file-etape) do control(att-etape)  
    forever  
end  
  
procedure control(e : etape);  
if test(etat, e) then execute(e)  
    else file-etape := inserer(e, file-etape)
```

Implémentation : Verrouillage en 2 phases

- **Etat du contrôleur** Verrous sur des données

- **Verrou partageable** (en lecture) LockR(A)

Un verrou en lecture sur A doit être obtenu par T avant la lecture de A par T : LockR(A) précède Read(A) dans T .

- **Verrou exclusif** (en écriture) LockW(A)

Un verrou en écriture sur A doit être obtenu par T avant l'écriture de A par T : LockW(A) précède Write(A) dans T .

- **Libération** de verrou Unlock(A)

Tout verrou doit être libéré avant la fin de la transaction (fin = commit ou abort).

- **2 phases** : Aucune action de verrouillage ne peut succéder à une libération de verrou

Verrouillage en 2 phases - 2 phase locking

R(A) W(A) R(B) W(B)

|

demande/libération des verrous – mode statique

↓

LockW(A,B) R(A) W(A) Unlock(A) R(B) W(B) Unlock(B)

R(A) W(A) R(B) W(B)

|

demande/libération des verrous – mode dynamique

↓

LockR(A) R(A) LockW(A) W(A) LockW(B) Unlock(A) R(B)
W(B) Unlock(B)

Verrouillage en deux phases strict

- Les verrous sont libérés **uniquement à la fin** de la transaction.

R(A) W(A) R(B) W(B)

|

demande/libération des verrous – mode dynamique strict

↓

LockR(A) R(A) LockW(A) W(A) LockR(B) R(B) LockW(B)
W(B) **Unlock(A,B)**-commit

- **strict** → annulation tjrs possible
pas de cascades d'annulation

Verrouillage en deux phases

- Les demandes et libérations de verrou sont insérées automatiquement par le système dans la transaction.

- **le test** (statique/dynamique)

Toute action est exécutée sauf si cette action est une demande de verrou et qu'un verrou exclusif est déjà posé sur la donnée par une autre transaction.

Verrouillage en deux phase

entrée :

T_1		LockW(A) R(A) W(A)
T_2		LockW(A) R(A) W(A) LockW(B) R(B) W(B) unLock(A,B)

LockW(B) R(B) W(B) unLock(A,B)		T_1
		T_2

sortie : ??

Gestion des interblocages

Maintenir un graphe des “attentes”

- Résoudre les interblocages

Annuler une (ou plusieurs) des transactions bloquantes

- prévention des interblocages

Horloge / seuil d'attente — priorité (estampille)

SQL2 – Niveaux d'isolation

- **Caractéristiques d'une transaction**

access mode, diagnostics size, isolation level

- **Isolation**

Niveau	Lecture non validée	Lecture non reproductible	Fantôme
Read Uncommitted	possible	possible	possible
Read Committed	non	possible	possible
Repeatable Read	non	non	possible
Serializable	non	non	non

Exercices - Concurrency et Verrouillage en deux phases

Construire les graphes de s erialisabilit e pour les ex ecutions suivantes. Indiquez les ex ecutions s erialisables et donnez leur ex ecution en s erie  equivalente dans ce cas. Quelles sont les ex ecutions  equivalentes ?

1.

T_1		R(A)	W(C)
T_2	W(A)	W(B)	
T_3		W(C)	R(B)

2.

T_1	R(A)		W(C)
T_2		W(B)	W(A)
T_3		R(B)	W(C)

3.

T_1		W(C)	R(A)
T_2		W(B)	W(A)
T_3	W(C)		R(B)

Exercices - Concurrency et Verrouillage en deux phases

Le programme suivant s'exécute dans un système de gestion de commandes pour les produits d'une entreprise. Il permet de commander une quantité donnée de produit en stock. Les paramètres du programme sont les références de la commande (c) et du produit (p) et la quantité demandée.

Commander(c, p, q)

lecture prix pp du produit p ;

si $q >$ stock de p alors Abort sinon

Mise à jour de stock de p Facturation de la commande c Commit

Quelles sont les transactions parmi les suivantes qui ont été produites par le programme ci-dessus ?

- R(x) R(y) Abort
- R(X) Abort
- R(X) W(Y) W(Z) Commit
- R(X) R(Y) W(Y) W(Z) Commit

Exercices - Concurrency and Two-Phase Locking

Three transactions execute at the same time: two orders for the same product and a price increase of the product. Show that the sequence of actions below is a concurrent execution of these transactions and explain. Is it serializable? What is the execution produced by two-phase locking, static, dynamic strict? Compare in each case the prices of the product applied for the two orders.

T_1	R(x) R(y)	W(y)	W(z) C
T_2	W(x)	C.	
T_3		R(x) R(y)	W(y) W(u) C

Exercices - Concurrency et Verrouillage en deux phases

Un contrôleur appliquant le protocole de verrouillage en deux phases strict reçoit la séquence d'actions suivantes. Que produit il en sortie ? On suppose que toute opération bloquée (en attente d'un verrou) est exécutée en priorité dès que le verrou est relâché par la transaction détentrice.

T_1	R(x)	W(y) W(x)	R(y) C
T_2	R(y)	W(y) C	
T_3	W(x)	R(y)	W(y) C