


Bases de données relationnelles



Concurrence

[HTTP://WWW.BDPEDIA.FR/SYSTEMES-RELATIONNELS/](http://www.bdpedia.fr/systemes-relationnels/)

Introduction



- Quand on développe un programme P accédant à une base de données, on effectue en général plus ou moins explicitement deux hypothèses:
 - P s'exécutera indépendamment de tout autre programme ou utilisateur
 - l'exécution de P se déroulera toujours intégralement
- BD = ressources accessibles simultanément à plusieurs utilisateurs
 - rechercher, créer, modifier ou détruire des informations
 - les accès simultanés à une même ressource sont dits concurrents
- Un programme ne s'exécute pas toujours jusqu'à son terme
 - l'arrêt du serveur de données
 - une erreur de programmation entraînant l'arrêt de l'application
 - la violation d'une contrainte amenant le système à rejeter les opérations demandées
 - une annulation décidée par l'utilisateur



Introduction

- Un concepteur d'application doit
 - Maîtriser la notion - très importante - de transaction
 - Réaliser l'impact des exécutions transactionnelles concurrentes sur les autres utilisateurs
 - Choisir un niveau d'isolation approprié
- Contrôle de concurrence



Objectif

- Plusieurs utilisateurs se partagent la même BD
- Un SGBD doit gérer les conflits qui peuvent se produire lorsque plusieurs utilisateurs manipulent simultanément les mêmes données
- la BD
 - Ne doit pas être mise dans un état incohérent
 - Régler les conflits lecture / écriture
 - Doit garder de très bonnes performances



Concept de transaction

- Une transaction est un fragment de programme dont l'exécution fait passer une BD d'un état cohérent à un autre état cohérent
- Concurrence
 - Permettre l'exécution simultanée d'un grand nombre de transactions



Concept de transaction

- Une transaction est une séquence d'opérations de lecture ou d'écriture, se terminant par
 - commit : instruction qui valide toutes les mises à jour
 - rollback : instruction qui annule toutes les mises à jour
- Une transaction est l'unité de traitement d'un SGBD



Concept de transaction

- Une transaction est le produit d'un échange entre un processus client et un processus serveur (SGBD)
- Le SGBD ne sait pas ce que fait l'application avec les données transmises
 - Il ne voit que la séquence des lectures et des écritures
- On peut effectuer une ou plusieurs transactions successives dans un même processus : elles sont dites sérielles
- En revanche, deux processus distincts engendrent des transactions concurrentes



Exemple

- Des clients réservent des places pour des spectacles

`Client (id_cl, nb_places_réservées, solde)`

- Des spectacles proposent des places à des clients.

`Spectacle (id_sp, nb_places_offertes, nb_places_prises, tarif)`

- Cette base est cohérente si le nombre de places prises est égal à la somme des places réservées

`Reservation (id_client, id_spectacle, nb_places)`



```
procedure Reservation (v_id_client INT, v_id_spectacle INT, nb_places INT)
-- Variables
v_client Client%ROWTYPE;
v_spectacle Spectacle%ROWTYPE;
v_places_libres INT;
v_places_reservees INT;
BEGIN
-- On recherche le spectacle
SELECT * INTO v_spectacle
FROM Spectacle WHERE id_spectacle=v_id_spectacle;

-- S'il reste assez de places: on effectue la reservation
IF (v_spectacle.nb_places_libres >= nb_places)
THEN
-- On recherche le client
SELECT * INTO v_client FROM Client WHERE id_client=v_id_client;

-- Calcul du transfert
v_places_libres := v_spectacle.nb_places_libres - nb_places;
v_places_reservees := v_client.nb_places_reservees + nb_places;

-- On diminue le nombre de places libres
UPDATE Spectacle SET nb_places_libres = v_places_libres
WHERE id_spectacle=v_id_spectacle;

-- On augmente le nombre de places reervees par le client
UPDATE Client SET nb_places_reservees=v_places_reservees
WHERE id_client = v_id_client;

-- Validation
commit;
ELSE
rollback;
END IF;
END;
```

Ce n'est pas du postgresSQL, PLPGSQL

Les transactions engendrées par Réservation



En s'exécutant, la procédure Réservation engendre des transactions. Exemples :

- $r[s_1]r[c_1]w[s_1]w[c_1]C$: on lit le spectacle s_1 , le client c_1 , puis on les met à jour tous les deux ;
- $r[s_1]r[c_2]w[s_1]w[c_2]C$: un autre client (c_2) réserve pour le même spectacle (s_1) ;
- $r[s_1]$: on lit le spectacle s_1 , et on s'arrête là (plus assez de places disponibles?)

Un même processus peut effectuer des transactions **en série** :

$$r_1[s_1]r_1[c_1]w_1[s_1]w_1[c_1]C_1r_2[s_1]r_2[c_2]w_2[s_1]w_2[c_2]C_2\cdots$$

- Quand plusieurs programmes clients sont actifs simultanément, les transactions engendrées s'effectuent en concurrence
- On obtient potentiellement un entrelacement des requêtes
- L'entrelacement de requêtes issues de transactions concurrentes peut engendrer des anomalies



Concept de transaction

- Une transaction est une suite d'opérations – une suite d'événements dont chacun peut être:
 - la lecture ou l'écriture d'une donnée
 - le verrouillage ou le déverrouillage d'une donnée
 - le démarrage ou l'arrêt (validation, annulation) de la transaction
- Une donnée est un fragment d'une BD
 - une valeur d'attribut, un n-uplet, une table...



Événements

événement	signification
start	démarrage de la transaction
read D	lecture d'une donnée D
write D	modification d'une donnée D
rollback	annulation de la transaction
commit	confirmation de la transaction



ACID : les propriétés d'une transaction

■ Atomicité

- une transaction doit s'exécuter en totalité, une exécution partielle est inacceptable
- Si elle est confirmée (commit) toutes les modifications qu'elle a effectuées sont enregistrées dans la BD et rendues visibles aux autres utilisateurs
- Si elle est interrompue (rollback) alors aucune de ces modifications n'est enregistrée dans la BD

Les opérations d'une transaction sont solidaires :
elles sont toutes validées, ou pas du tout



ACID : les propriétés d'une transaction

■ Cohérence

- Une transaction fait passer une BD d'un état cohérent à un autre état cohérent
- Un état cohérent est un état dans lequel les contraintes d'intégrité sont vérifiées
- Pendant la transaction, l'état peut être incohérent



ACID : les propriétés d'une transaction

■ Isolation

- Une transaction se déroule sans être perturbée par les effets des transactions concurrentes
 - tout se passe comme si elle se déroulait seule
- Idée: exécution concurrente de transactions équivalente à une exécution en série (non concurrente)



ACID : les propriétés d'une transaction

■ Durabilité

- Les effets d'une transaction validée sont permanents
- Une fois qu'une transaction a été confirmée le SGBD garantit qu'aucune modification qu'elle a effectuée ne sera perdue quels que soient les accidents qui surviendront
 - interruption, pannes du système d'exploitation, «crash» de disque, etc.



Concurrence

- Plusieurs transactions s'exécutent en même temps
 - Le système exécute les opérations en séquence
 - Entrelacement des opérations de plusieurs transactions
 - tout entrelacement n'est pas acceptable

→ Exécution correcte en respectant les propriétés ACID



Les problèmes de concurrence

- Perte de mises à jour
- Lecture impropre
- Lecture non reproductible

Mise à jour calculée à partir d'une valeur périmée de donnée



Perte de mises à jour

T_1	T_2	BD
		$A = 10$
read A		
	read A	
$A = A + 10$		
write A		$A = 20$
	$A = A + 50$	
	write A	$A = 60$

Les deux transactions lisent la même donnée
puis les deux transactions l'écrivent
→ Une des deux mises à jour est perdue



lecture impropre Données incohérentes

T_1	T_2	BD
		$A + B = 200$
		$A = 120$ $B = 80$
read A		
$A = A - 50$		
write A		$A = 70$
	read A	
	read B	
	display A + B (150 est affiché)	
read B		
$B = B + 50$		
write B		$B = 130$

T1 est cohérente vis à vis de la contrainte
Cependant l'ordre des opérations est tel que l'affichage réalisée par T2 est incohérent vis à vis de la contrainte

Lecture impropre Données non validées

T_1	T_2	BD
		$A = 50$
	$A = 70$	
	write A	$A = 70$
read A (70 est lu)		
	rollback (La valeur initiale de A est restaurée)	$A = 50$

T_1 a lu une valeur de A incorrecte car tout doit se passer comme si T_2 n'avait pas eu lieu.

Lecture non reproductible

Deux lectures d'une même donnée dans une même transaction conduisent à deux valeurs différentes

T_2 qui ne modifie pas A devrait toujours Lire la même valeur pour A

T_1	T_2	BD
		$A = 10$
	read A (10 est lu)	
$A = 20$		
write A		$A = 20$
	read A (20 est lu)	



Objet fantôme

T_1	T_2	BD
		$E = \{1, 2, 3\}$
display card(E) 3 est affiché		
	insert 4 into E	$E = \{1, 2, 3, 4\}$
display card(E) 4 est affiché		



Contrôle de concurrence

- Réordonnancement des opérations d'un ensemble de transactions (→ en retarder)
- Exécution sérialisable d'un ensemble de transactions
 - **équivalente** à une exécution en série quelconque des transactions (Appliquer systématiquement une exécution en série serait trop coûteux)

l'ordre des opérations dans chaque transaction doit être respecté



Définition de la sérialisabilité

- L'exécution d'un ensemble de transactions est **en série** si
 - pour tout couple de transactions, tous les événements de l'une précèdent tous les événements de l'autre
- Deux exécutions d'un même ensemble de transactions sont **équivalentes** ssi:
 - elles sont constituées des mêmes événements
 - elles produisent le même état final de la BD et les mêmes résultats pour les transactions
 - Les lectures produisent les mêmes résultats
 - les écritures sont réalisées dans le même ordre

Une exécution concurrente d'un ensemble de transactions est dite **sérialisable** ssi il existe une exécution en série équivalente ...

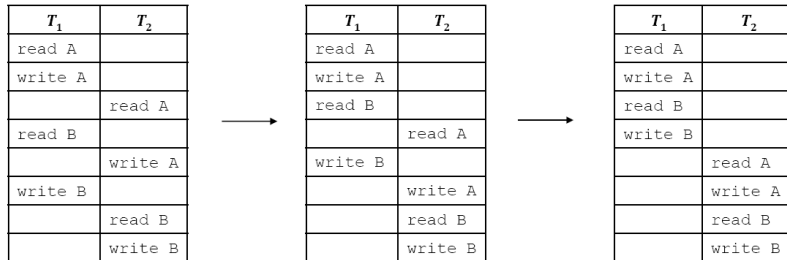


Définition de la sérialisabilité

- Une exécution concurrente est sérialisable si elle peut être transformée en une exécution en série équivalente **par une suite de permutations d'opérations non conflictuelles**
- 2 opérations sont conflictuelles si
 - elles appartiennent à deux transactions différentes
 - elles ne sont pas permutable
 - elles portent sur la même donnée
 - l'une des deux opérations est une opération d'écriture



Exemple – Une exécution sérialisable



Contrôle par verrouillage

- *Prévenir les conflits – technique pessimiste*
- **Verrouillage** des objets en lecture/écriture
 - Avant de lire ou écrire une donnée, une transaction demande un verrou sur cette donnée pour interdire aux autres transactions d'y accéder
 - Si ce verrou ne peut être obtenu, parce qu'une autre transaction en possède un, la transaction demandeuse est **mise en attente**



Granularité de verrouillage

- On peut verrouiller
 - Un n-uplet (toutes ses valeurs)
 - Une page (tous ses n-uplets)
 - Une table (tous ses n-uplets)
 - La BD (toutes ses tables)



Modes de verrouillage

- Mode partagé (S) - lecture
 - Verrou demandé avant de lire une donnée
- Mode exclusif (X) - écriture
 - Verrou demandé avant de modifier une donnée

Opération	Signification
lock m D	demande d'un verrou en mode m sur la donnée D
unlock D	déverrouillage d'une donnée D



Règles régissant les modes S et X

- Un **verrou partagé (S)** ne peut être obtenu sur une donnée que si les verrous déjà placés sur cette donnée sont eux même partagés
- Un **verrou exclusif (X)** peut être obtenu sur une donnée si aucun verrou n'est déjà placé sur cette donnée



Verrouillage à deux phases

- Une transaction est dite **bien formée** si
 - Elle obtient un verrou sur une donnée avant de la lire ou de l'écrire
 - Elle libère tous ses verrous avant de se terminer
- Une transaction est dite **à deux phases** si
 - Elle est bien formée
 - Après avoir relâché un verrou, elle n'en acquière plus
 - Une Transaction relâche ses verrous après les avoir tous acquis

L'exécution d'un ensemble de transactions à 2 phases est sérialisable (dans l'ordre du début de leur phase de libération)



Plus de perte de mise à jour

T_1	T_2	BD
		$A = 10$
lock X A		
read A		
	lock X A	
$A = A + 10$	attente	
write A	attente	$A = 20$
unlock A	attente	
	read A	
	$A = A + 50$	
	write A	$A = 70$
	unlock A	

Le verrouillage de A par T1 oblige T2 à attendre que T1 ait terminé sa mise à jour de A avant d'effectuer la sienne. Il n'y a pas de perte de mise à jour.



Plus de lecture impropre

T1	T2	BD
		Contrainte $A + B = 200$ $A = 120$ $B = 80$
Lock X A		
Read A		
$A = A - 50$		
Write A		$A = 70$
	Lock S A	
Lock X B	attente	
Read B	attente	
$B = B + 50$	attente	
Write B	attente	$B = 130$

T1	T2	BD
		Contrainte $A + B = 200$ $A = 70$ $B = 130$
Unlock A	attente	
	Read A	
Unlock B		
	Lock S B	
	Read B	
	Display A+B (200 est affiché)	
	Unlock B	

Le verrouillage de A par T1 empêche T2 de lire A avant que T1 n'ait terminé sa mise à jour de A et de B. T2 affiche donc une valeur cohérente de A+B



Plus de lecture non reproductible

Le verrouillage de A par T1 empêche T2 de modifier A entre les 2 lectures

T ₁	T ₂	BD
		A = 10
	lock S A	
	read A (10 est lu)	
lock X A		
attente	read A (10 est lu)	
attente	unlock A	
A = 20		
write A		A = 20



Nuplets fantômes

Le verrouillage des *n*-uplets ne résoud pas le problème.

T ₁	T ₂
SELECT COUNT(*) FROM livre WHERE année = 2003; (réponse <i>n</i>) SELECT COUNT(*) FROM livre WHERE année = 2003; (réponse <i>n</i> + 1) COMMIT	INSERT INTO livre VALUES ("Les BD", 2003); COMMIT

Chaque *n*-uplet de la table *livre* lu par T₁ est verrouillé en lecture, mais cela n'a pas d'influence sur la création d'un nouveau *n*-uplet par T₂.

Nuplets fantômes

Le verrouillage des tables empêche l'apparition de n -uplets fantômes.

T_1	T_2
LOCK S livre SELECT COUNT(*) FROM livre WHERE année = 2003; (réponse n) SELECT COUNT(*) FROM livre WHERE année = 2003; (réponse n) COMMIT	LOCK X livre attente attente attente attente attente INSERT INTO livre VALUES ("Les BD", 2003); COMMIT

Problème de verrouillage

- Verrou mortel : interblocage
 - risques de circuit d'attentes entre transactions



T1 lock X A	T2 lock X B
T1 lock S B	T2 lock S A
Attente	T2 lock S A
attente	attente

Un graphe d'attente est un graphe orienté dont les nœuds sont les transactions et les arcs les attentes entre transactions



Résolution de l'interblocage

- En les évitant (prévention)
 - On accorde par exemple à une transaction tous les verrous dont elle a besoin avant son démarrage
- En les détectant
 - On inspecte régulièrement le graphe d'attente
 - Si blocage (cycle), on défait l'un des deux T (la moins coûteuse) et on la relance plus tard
 - On annule une T si le temps d'attente (time out) dépasse un seuil et on la relance plus tard
 - Paramétrage fin



Bilan verrouillage

- Approche pessimiste
 - prévient les conflits
 - assez coûteuse
 - assez complexe
- Approche retenue
 - dans tous les SGBD industriels
- Difficile de faire mieux



SQL la norme

- Une transaction est une suite de commandes SQL
- Une transaction est démarrée par un agent lorsqu'il exécute une commande SQL et qu'il n'y a pas de transactions en cours
- Une transaction est terminée explicitement par une commande COMMIT ou une commande ROLLBACK
- Deux transactions ne peuvent pas être imbriquées
 - un agent ne peut pas démarrer une transaction si sa transaction courante n'est pas terminée



Les 4 niveaux d'isolation

Plus le niveau est permissif, plus l'exécution est fluide, plus les anomalies sont possibles.

Plus le niveau est strict, plus l'exécution risque de rencontrer des blocages, moins les anomalies sont possibles.

Les 4 niveaux d'isolation

- READ UNCOMMITTED
 - Il peut y avoir des pertes de mise à jour, des lectures impropres, des lectures non reproductibles et des objets fantômes
 - Il n'y a pas d'attente en lecture
- READ COMMITTED
 - Il ne peut pas y avoir de lectures impropres
 - Il peut y avoir des pertes de mise à jour, des lectures non reproductibles et des objets fantômes
 - Les transactions n'ont accès qu'aux données produites par les transactions confirmées
- REPEATABLE READ
 - Il ne peut pas y avoir de lectures impropres, ni de lectures non reproductibles
 - Il peut y avoir des pertes de mise à jour et des objets fantômes
- SERIALIZABLE
 - L'isolation est totale

A RETENIR

- READ COMMITTED
 - **une requête accède à l'état de la base au moment où la requête est exécutée**
 - Pas de lecture sale, car une donnée en cours de modification ne fait pas partie de l'état de la base
 - Assez fluide, mais autorise beaucoup d'anomalies
- REPEATABLE READ
 - **une requête accède à l'état de la base au moment où la transaction a débuté**
 - Pas de lecture sale, pas de lecture non répétable : les requêtes accèdent toujours au même état de la base
 - Autorise les mises à jour perdues et les tuples fantômes
- SERIALIZABLE
 - garantit l'isolation totale, et donc la cohérence de la base
 - risque non négligeable de rejet de l'une des transactions



Niveau d'isolation SQL

	Lectures sales	Lectures non répétables	Tuples fantômes
read uncommitted	Possible	Possible	Possible
read committed	Impossible	Possible	Possible
repeatable read	Impossible	Impossible	Possible
serializable	Impossible	Impossible	Impossible



Mode d'exécution d'une transaction

Le mode d'exécution d'une transaction est spécifié par la commande `SET TRANSACTION` qui la précède immédiatement :

`SET TRANSACTION ISOLATION LEVEL <niveau>`

```
niveau d'isolation ::=  
    READ UNCOMMITTED  
  | READ COMMITTED  
  | REPEATABLE READ  
  | SERIALIZABLE
```

Attention ! la commande `SET TRANSACTION` n'est pas un début de transaction et elle ne peut pas être utilisée si une transaction est en cours.



Atomicité et durabilité

- Le respect de l'atomicité peut impliquer de défaire les effets d'une transaction lorsque celle-ci a été abandonnée
- Le respect de la durabilité implique que le SGBD doit être capable de remettre la base de données en état après une panne
 - les mises à jour faites par une transaction non confirmée avant la panne doivent être défaites
- C'est le gestionnaire de **reprise** qui assure cette tâche