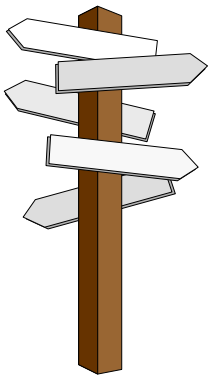
UPX
IUP MIAGE
UNIVERSITE PARIS X NANTERRE

Cours n°8
Transactions et accès concurrents

Chantal Reynaud

Université Paris X - Nanterre UFR SEGMI - IUP MIAGE



Plan

I. Introduction et problématique

II. Contrôle de concurrence

III. Gestion des accès concurrents et SQL

Partie I. Introduction et problématique

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

I. Notion de transaction

- Une ***transaction*** est un ensemble de modifications de la base qui forme un tout indivisible. Une transaction est composée d'un ensemble d'opérations élémentaires qui transforment un état cohérent en un autre état cohérent, sans nécessairement préserver la cohérence de toutes les étapes intermédiaires.

- Un système qui assure une ***gestion des transactions*** doit apporter la garantie que, si la transaction exécute certaines mises à jour et qu'une défaillance survient ensuite avant que la transaction n'ait atteint le point de terminaison prévu, alors toutes les mises à jour doivent être annulées.

Opérations de contrôle des transactions

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

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

<u>Exemple :</u>

opérations $r_i[x]$, $w_i[x]$, R_i, C_i (R signifie Rollback, C signifie Commit)

T_1 : $r_1[x]$ $w_1[x]$ $\mathbf{C}_1$ (crédit)
T_2 : $r_2[x]$ $w_2[x]$ $r_2[y]$ $w_2[y]$ $\mathbf{C}_2$ (transfert)
T_3 : $r_3[x]$ $\mathbf{R}_3$ (transfert impossible)

} correct

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

Les propriétés ACID

- **A**tomicité

Les transactions doivent s'exécuter en totalité. Une exécution partielle est inacceptable.

- **C**ohérence

Les transactions transforment un état cohérent de la base en un autre état cohérent sans nécessairement préserver la cohérence de tous les états intermédiaires

- **I**solation

Même si plusieurs transactions s'exécutent de façon concurrente, toutes les mises à jour d'une transaction donnée seront cachées du reste jusqu'à ce que la transaction soit validée.

- **D**urabilité

Une fois une transaction validée, ses mises à jour doivent perdurées, même si une défaillance du système survient.

II. Transactions concurrentes

- Deux transactions sont concurrentes si elles accèdent en même temps aux mêmes données.
- Pour des raisons d'efficacité, il est nécessaire d'autoriser l'exécution concurrente des transactions. Tout le problème réside alors dans le fait qu'il faut gérer ces exécutions concurrentes pour qu'elles ne conduisent pas à une base incohérente.
- C'est l'***entrelacement*** non contrôlé des opérations de deux transactions correctes qui produit un résultat global incorrect.
- Les opérations sont toujours ***exécutées en séquence***.

Problèmes de concurrence

- Le problème de perte de mise à jour

<u>Exemple :</u>

$T_1 = r_1 [x] w_1 [x] C_1$ (crédit x de 100)
$T_2 = r_2 [x] w_2 [x] C_2$ (crédit x de 50)
au début, x = 200

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

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

Problème de cohérence même si l'isolation est respectée.

Problèmes de concurrence

- Le problème des dépendances non validées

<u>Exemple :</u>

$T_1 = r_1 [x] w_1 [x] R_1$ (crédit x de 100)
$T_2 = r_2 [x] w_2 [x] C_2$ (crédit x de 50)
au début, x = 200

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

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

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

Problèmes de concurrence

- Le problème des analyses incohérentes

T_1 = transfert de x → y de 50
T_2 = calcul dans z de la somme x + y
au début, x = 200, y = 100

$T_1 = r_1[x]\ w_1[x]\ r_1[y]\ w_1[y]\ C_1$
$T_2 = r_2[x]\ r_2[y]\ w_2[z]\ C_2$

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

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

Problème de cohérence à cause de l'isolation.

III. Le problème des annulations

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

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

Donc, une transaction validée risque d'être annulée !

• On imposera alors que les exécutions concurrentes satisfassent les propriétés suivantes : recouvrabilité, éviter les annulations en cascade, exécution stricte.

Propriété de recouvrabilité

- Exemple :

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

R_1 oblige l'annulation de T_2. Or, ce n'est pas possible car T_2 a été validée. On parle d'exécution *non recouvrable*.

Une exécution non recouvrable est à éviter absolument car elle introduit un conflit insoluble entre les annulations d'une transaction et les validations d'une autre.

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

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

Annulations en cascade

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

Annulations en cascade

$w_1[x]$ $r_2[x]$ $w_2[y]$ R_1 → R_1 oblige R_2

- Solution : T_2 ne doit lire qu'à partir de transactions validées

→ *retardement des lectures*

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

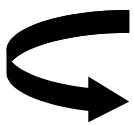
Exécution stricte

- Exemple :

$w_1[x,2]$ $w_2[x,3]$ R_1 R_2
au début, x=1
Image avant ($w_1[x]$) =1, Image avant ($w_2[x]$) =2
R_1 restaure x = 1: erreur R_2 restaure x = 2 : erreur

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

→ *retardement des lectures et des écritures*



Deux transactions ne peuvent pas accéder simultanément en écriture à la même donnée, sous peine de ne pouvoir utiliser la technique de récupération de l'image avant.

Partie II. Le contrôle de concurrence

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

I. Théorie de la sérialisabilité

• Objectif : produire une exécution sérialisable des transactions, c'est-à-dire équivalente à une exécution en série quelconque des transactions.

<u>Exemple d'exécutions en série :</u>

Au début x = 200. T_1 = crédit de 100; T_2 = crédit de 50;

$H_1 = T_1\ T_2 = r_1\ [x]_{x=200}\ w_1\ [x]_{x=300}\ C_1\ r_2\ [x]_{x=300}\ w_2\ [x]_{x=350}\ C_2$

$H_2 = T_2\ T_1 = r_2\ [x]_{x=200}\ w_2\ [x]_{x=250}\ C_2\ r_1\ [x]_{x=300}\ w_1\ [x]_{x=350}\ C_1$

• Equivalence de deux exécutions (histoires)

- avoir les mêmes transactions et les mêmes opérations
- produire le même effet sur la BD (écritures)
- produire le même effet dans les transactions (lectures)

➡ H_1 non équivalent à H_2 (conditions 2 et 3 non respectées)

Théorie de la sérialisabilité

- Conflit

Def : $p_i[x]$ et $q_j[y]$ sont en conflit $\Leftrightarrow$

- $i \neq j$, $x = y$ (transactions différentes, même enregistrement)
- $p_i[x]$ $q_j[x]$ n'a pas le même effet que $q_j[x]$ $p_i[x]$

- Exemples :

$w_i[x]$ - $w_j[x]$, $r_i[x]$ - $w_j[x]$, $w_i[x]$ - $r_j[x]$
Seul le couple $r_i[x]$ - $r_j[x]$ n'est pas en conflit

Théorie de la sérialisabilité

- Critère d'équivalence utilisé

- avoir les mêmes transactions et les mêmes opérations
- avoir le même ordre des opérations conflictuelles dans les transactions non annulées

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

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

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

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

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

Donc H_2 est équivalent à H_1.

Théorie de la sérialisabilité

- Théorème de sérialisabilité

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

SG = graphe de sérialisation d'une exécution H :
nœuds = transactions T validées dans H
Arcs : si p et q conflictuelles, $p \in T_i$, $q \in T_j$,
p avant q $\Rightarrow$ arc $T_i \rightarrow T_j$

- Exemple de graphe de sérialisation pour H_1 et H_2 :

$$T_1 \rightleftarrows T_2$$

H_1 et H_2 ne sont donc pas sérialisables.

II. Contrôle par verrouillage à deux phases

- Théorème

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

- Protocole

1) Avant d'agir sur un objet, une transaction doit obtenir un verrou sur cet objet.
2) Après l'abandon d'un verrou, une transaction ne doit plus jamais pouvoir obtenir de verrous.
Une transaction obéissant à ce protocole a donc deux phases, une phase d'acquisition de verrous et une phase d'abandon de verrous.
Le respect du protocole est assuré par un module dit *scheduler* qui reçoit les opérations émises par les transactions et les traite.

Contrôle par verrouillage à deux phases

- Algorithme

1) Le scheduler reçoit $p_i[x]$ et consulte le verrou déjà posé sur x, $ql_j[x]$, 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
2) Un verrou pour $p_i[x]$ n'est jamais relâché avant la confirmation de l'exécution par le gestionnaire des données
3) Dès que T_i relâche un verrou, elle ne peut plus en obtenir d'autre.

- Théorème

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

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

Contrôle par verrouillage à deux phases

- Exemple :

T_1 : r_1 [x] w_1 [y] C_1
T_2 : r_2 [y] w_2 [y] C_2
Ordre de réception : r_1 [x] r_2 [y] w_1 [y] C_1 w_2 [y] C_2

r_1 [x] exécutée
r_2 [y] exécutée
w_1 [y] retardée à cause de r_2 [y] et tout le reste de T_1 va être bloqué
C_1 bloqué
w_2 [y] exécutée
C_2 relâche les verrous sur y
w_1 [y] exécutée
C_1 exécutée

Exécution correcte : r_1 [x] r_2 [y] w_2 [y] C_2 w_1 [y] C_1
Conflits : r_2 [y] - w_1 [y] ; w_2 [y] - w_1 [y]
Pas de cycle, donc H sérialisable

Contrôle par verrouillage à deux phases

- Cette stratégie peut parfois conduire à un interblocage (cas d'un accès aux mêmes ressources mais dans un ordre différent)
- Exemple :

T_1 : r_1 [x] w_1 [y] C_1
T_2 : w_2 [y] w_2 [x] C_2
Ordre de réception : r_1 [x] w_2 [y] w_2 [x] w_1 [y]

T_1 obtient un verrou pour r_1[x], T_2 pour w_2[y]
w_2[x] attend r_1[x], w_1[y] attend w_2[y]
interblocage de T_1 et de T_2.

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

Partie III. Gestion des accès concurrents et SQL

Il est possible en SQL de choisir explicitement le niveau de protection que l'on souhaite obtenir contre les incohérences résultant de la concurrence d'accès.

Comportement par défaut : assurer la sérialisabilité et la recouvrabilité stricte

Options possibles :

1) On spécifie qu'une transaction ne fera que des lectures
SET TRANSACTION READ ONLY;
2) Option par défaut : une transaction peut lire et écrire
SET TRANSACTION READ WRITE;

SQL2 et les propriétés des exécutions concurrentes

La norme SQL2 spécifie que ces exécutions doivent être sérialisables (mode par défaut). Un verrouillage strict doit alors être assuré par le SGBD.

SQL2 propose des options moins fortes :

SET TRANSACTION ISOLATION LEVEL *option*

Liste des options :

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