Chapitre VI

Sécurité

Chapitre VI

Sécurité

Contrôle de l'accès à la base

- Le contrôle de l'accès à la base est effectué en associant à chaque utilisateur
  - un nom de login
  - un mot de passe

  Chaque utilisateur a des privilèges d'accès à la base : droit ou non
  - de créer des tables ou des vues
  - de lire ou de modifier des tables ou des vues
  - ...

Propriétaire des données

- Une table (et les données qu'elle contient) appartient à celui qui l'a créé
- Le propriétaire d'une table peut donner à d'autres le droit de travailler avec sa table
- Les vues permettent d'affiner les droits que l'on donne sur ses propres données : on peut donner des droits sur des vues et pas sur les tables sous-jacentes

Accorder des droits sur des objets

Définition

GRANT privilège [(colonne,...)]ON table/vue
TO utilisateur
[WITH GRANT OPTION]

- L'utilisateur qui reçoit le privilège pourra le donner à d'autres utilisateurs
- Les privilèges :
  - SELECT
  - INSERT
  - UPDATE [(col1, col2,...)]
  - DELETE
  - ALTER
  - ALL PRIVILEGES

Accorder des droits (exemples)

Exemple

```
grant select on emp to clement;
grant select, update on emp to clement, chatel;
grant select on emp to public;
```

Changer son mot de passe :

Exemple

```
grant connect to bibi identified by motDePasse;
```

Reprendre les droits

Définition

REVOKE privilège ON table/vue FROM utilisateur

Chapitre VI

Sécurité

Gestion des accès concurrents

- Un SGBD est prévu pour travailler avec de nombreux utilisateurs/transactions
- ... qui peuvent consulter et modifier en même temps les mêmes données

Quelques questions

- A quel moment les modifications sont-elles **vues** par les autres transactions ?
- Que se passe-t-il lorsque plusieurs transactions veulent modifier les **mêmes** données ?
- Un ordre SQL (moyenne des salaires, par exemple) tient-il compte des modifications apportées par les autres transactions **validées** pendant son exécution ?

Problèmes liés aux accès concurrents

- Il peut se produire des pertes de données quand plusieurs processus veulent modifier les mêmes données en même temps
- Les principaux cas d'école sont :
  - mise à jour perdue
  - lecture inconsistante
  - lecture non reproductible
  - ligne fantôme

Mise à jour perdue

$S = 500$

Temps

Transaction T1	Transaction T2
s = Lire S	
	s = Lire S

Mise à jour perdue

$S = 1500$

Temps

Transaction T1	Transaction T2
s = Lire S	
	s = Lire S
s = s + 1000	

Mise à jour perdue

$S = 2500$

Temps

Transaction T1	Transaction T2
s = Lire S	
	s = Lire S
s = s + 1000	
	s = s + 2000

Mise à jour perdue

$S = 1500$

	Temps
Transaction T1	Transaction T2
s = Lire S	
	s = Lire S
s = s + 1000	
	s = s + 2000
Enregistrer s	

Mise à jour perdue

$S = 2500$

Temps

Transaction T1	Transaction T2
s = Lire S	
	s = Lire S
s = s + 1000	
	s = s + 2000
Enregistrer s	
	Enregistrer s

Éviter les mises à jour perdues

Temps

Transaction T1	Transaction T2
Bloquer S	
s = Lire S	
	s = Lire S; attente ...
s = s + 1000	
Enregistrer s	
Débloquer S	... s = Lire S
	s = s + 2000
	Enregistrer s

Problèmes liés aux blocages : interblocage

Temps

Transaction T1	Transaction T2
Bloquer A	
	Bloquer B
Bloquer B; Attente ...	
	Bloquer A; Attente ...

Lecture inconsistante

Temps

Transaction T1	Transaction T2
V = 100	
	v = Lire V (100)
ROLLBACK	
	Travaille avec v=100

Ce cas n'arrive pas si les modifications ne sont visibles par les autres qu'après un COMMIT.

Lecture inconsistante

Temps

Transaction T1	Transaction T2
Lire V	
	V = V + 100
	COMMIT
Lire V	

Pour éviter cela, T1 devrait bloquer les données lues (V)

Lignes fantômes

- Ce problème survient quand une transaction n'a pas perçu la création d'une ligne par une autre transaction
- Par exemple, une transaction lit d'abord le nombre d'employés et lance ensuite la lecture d'informations sur chacun de ces employés
- Si une autre transaction a ajouté des employés, le nombre lu auparavant ne correspond plus à la réalité

Lignes fantômes

Temps

Transaction T1	Transaction T2
Récupère le nombre d'employés du dept 10 (35)	
	Ajoute un employé au dept 10
	COMMIT
Récupère les informations sur les employés du dept 10 (36 lignes)	

Pour éviter cela, T1 devrait bloquer la table des employés au début de la transaction (pas possible de bloquer des lignes qui n'existent pas !)

Transactions sérialisables

- On vient de voir que l'exécution de transactions **entrelacées** peut provoquer des pertes de données ou de cohérence
- Pour éviter ces problèmes, le SGBD doit s'arranger pour que l'exécution de plusieurs transactions entrelacées fournisse les mêmes résultats que l'exécution des mêmes transactions **les unes à la suite des autres** (dans un ordre quelconque)

Moyens de sérialiser

- Le moyen le plus courant s'appelle le **verrouillage à 2 phases** :
  - on doit bloquer un objet avant d'agir sur lui (lecture ou écriture)
  - on ne peut plus faire de blocage après avoir débloquer un objet
- On a donc 2 phases :
  1. acquisitions des verrous
  2. abandons des verrous (en pratique, au COMMIT ou ROLLBACK)

Inter-blocage avec le verrouillage à 2 phases

- Les situations d'entrelacement des transactions qui auraient provoqué des problèmes vont se traduire par des attentes ou des inter-blocages
- En cas d'inter-blocage, des transactions sont annulées et redémarrées

Estampillage

- L'estampillage est une autre stratégie que le verrouillage à 2 phases pour assurer la sérialisation des transactions
- L'ancienneté des transactions est repérée par une estampille donnée à la création de la transaction (un nombre incrémenté à chaque attribution)
- Chaque donnée accédée par une transaction reçoit l'estampille de cette transaction
- Les algorithmes qui utilisent l'estampillage assurent que l'exécution concurrente des transactions sera équivalente à l'exécution séquentielle de ces transactions dans l'ordre de leur estampille

Idée pour les algorithmes d'estampillage

- Une transaction T ne peut accéder à une donnée si cette donnée a déjà été accédée par une transaction plus jeune (donc d'estampille plus grande que celle de T)
- La transaction « trop vieille » T est annulée si elle s'est faite doubler par une transaction plus jeune
- Elle est relancée avec une estampille plus grande ; plus jeune, elle a plus de chance de pouvoir accéder à la donnée en passant « après »

Résultat de l'estampillage

- Cet algorithme est trop restrictif et ne va autoriser que peu d'exécutions parallèles des transactions
- La transaction redémarrée va peut-être entrer en conflit avec une autre transaction plus jeune
- D'autres algorithmes d'estampillage moins simplistes permettent d'améliorer le parallélisme, en particulier en distinguant les accès en lecture et les accès en écriture
- Malgré tout, ce type de traitement est souvent trop restrictif et nuit à la concurrence

Autres stratégies

- Pour améliorer les performances dans des situations de forte concurrence, les SGBD offrent la possibilité d'être plus permissif et de ne pas sérialiser les transactions

Niveaux d'isolation des transactions (InnoDB)

Définition

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE | READ COMMITED | READ UNCOMMITTED |
REPEATABLE READ

SERIALIZABLE : les transactions s'exécutent totalement isolées des autres transactions et comme si elles s'exécutaient les unes après les autres

Niveaux d'isolation des transactions (InnoDB)

Définition

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE | READ COMMITED | READ UNCOMMITTED |
REPEATABLE READ

READ COMMITED : les transactions ne voient les modifications des autres transactions qu'après les commit ; empêche les lectures inconsistantes mais pas les lectures non répétitives ni les lignes fantômes C'est l'isolation par défaut d'Oracle et de beaucoup de SGBD car c'est un bon compromis entre sécurité et performances

Niveaux d'isolation des transactions (InnoDB)

Définition

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE | READ COMMITED | READ UNCOMMITTED |
REPEATABLE READ

READ UNCOMMITED : les transactions voient les modifications avant même le commit

Niveaux d'isolation des transactions (InnoDB)

Définition

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE | READ COMMITED | READ UNCOMMITTED |
REPEATABLE READ

REPEATABLE READ : empêche les lectures non répétitives mais pas les lignes fantômes

En résumé...

Niveaux par isolation décroissante :

- SERIALIZABLE (souhaitable, mais coûteux car provoque des blocages de tables)
- REPEATABLE READ
- READ COMMITED (souvent un bon compromis)
- READ UNCOMMITED (très rarement utilisé)

Traitement des accès concurrents par les SGBD

- Le but : favoriser au maximum les accès concurrents pour permettre l'exécution du plus grand nombre de transactions dans un temps donné (argument commercial important)
- Tous les SGBDs n'ont pas les mêmes solutions pour atteindre ce but

Durée des blocages

Un blocage n'a lieu que le temps d'une transaction

Accès concurrents

- Pas de verrou pour effectuer une lecture
- Les lectures ne sont pas bloquées par les écritures, et vice-versa
- Une transaction est bloquée si elle veut modifier des données qui sont en cours de modification par une autre transaction
- La granularité minimum de blocage est la ligne (dépand du moteur (InnoBD, MyISAM, ...)

Autre façon de traiter les accès concurrents

- Beaucoup de SGBD bloquent les données **lues**
  - ce blocage n'interdit pas leur lecture par d'autres transactions mais interdit leur modification

  Une écriture bloque les données **modifiées**
  - elles ne peuvent être modifiées par d'autres transactions (comme avec Oracle)
  - mais elles ne peuvent pas non plus être lues par d'autres transactions

Blocages explicites et implicites

- Des blocages sont effectués implicitement par certaines commandes (en particulier UPDATE, INSERT et DELETE)
- Si le comportement par défaut du SGBD ne convient pas pour un traitement, on peut effectuer des blocages explicites des lignes ou des tables
- Les inter-blocages sont détectés par le SGBD qui annule un des ordres qui a provoqué l'inter-blocage

Granularité des blocages

- 2 types de blocages :
  - au niveau d'une table
  - au niveau d'une ligne
- Dans certains SGBDs le blocage d'une ligne implique le blocage de toute la page/bloc (plusieurs lignes) qui contient la ligne
- Certains SGBD transforment de trop nombreux blocages de lignes d'une table en un blocage de toute la table

Réponses à quelques questions

- A quel moment les modifications sont-elles vues par les autres transactions ?
  après la validation de la transaction (COMMIT)
- Que se passe-t-il lorsque plusieurs transactions veulent modifier les mêmes données ?
  blocages implicites du SGBD $\Rightarrow$ attente

Réponses à quelques questions

- Quand un ordre SQL tient compte de plusieurs lignes (moyenne des salaires, par exemple), cet ordre tient-il compte des modifications apportées par les autres transactions pendant l'exécution de l'ordre ?
  Ca dépend du SGBD. Avec MySQL, non