Quatrième Partie

Transactions, utilisateurs et privilèges

Transactions
Sommaire

- Transactions
  - Opérations élémentaires
  - Concurrence d’accès
- Utilisateurs et privilèges

Transactions
Définition

- Une transaction est une suite d'opérations qui forme un tout indivisible, donc :
  - soit entièrement réalisée
  - soit entièrement ignorée

- Avant la fin d'une transaction les modifications ne sont pas visibles des autres transactions -> « bac à sable »

- Les transactions jouent un rôle majeur dans :
  - la gestion des accès concurrents
  - la reprise sur panne
  - la réalisation matérielle des opérations
  - ...

- (Note : la notion de transaction est une des rares notions de BdD reprise dans divers autres domaines de l'informatique : filesystems, mémoires transactionnelles, etc.)

Transactions
Opérations élémentaires

- En général, une transaction débute (implicitement) par la première instruction DML
- La fin d'une transaction peut prendre deux formes :
  - `COMMIT` : les modifications apportées deviennent effectives, et sont rendues visibles aux autres transactions
  - `ROLLBACK` : les modifications apportées sont annulées, on revient à l'état de la base en début de transaction
- En général une instruction DDL effectue un `COMMIT` implicite de la transaction en cours, et constitue une transaction à elle seule
- La fin d'une connexion termine la transaction en cours : fin normale = `COMMIT`, fin anormale = `ROLLBACK`
- Éviter absolument les fins implicites

Transactions

Opérations élémentaires (suite)

- Certains systèmes/clients/bibliothèques/... Utilisent `l'AUTOCOMMIT` (une instruction = une transaction)
- Exemples :
  - JDBC (client)
  - MySQL (serveur)
  - ...
- Pratique (?), mais extrêmement dangereux
- Le découpage en transaction fait partie intégrante de l'analyse des traitements

Transactions
Opérations élémentaires (suite)

- Un point de sauvegarde est une étape intermédiaire, nommée `SAVEPOINT nom`
- On peut annuler la dernière partie d'une transaction `ROLLBACK TO SAVEPOINT nom`
- `ROLLBACK TO SAVEPOINT` a plusieurs effets :
  - annule les modification effectuées depuis le `SAVEPOINT` correspondant
  - supprime tous les points de sauvegardes depuis le `SAVEPOINT` correspondant
  - libère les verrous
- Attention à l'interaction avec les instructions DDL
- Vraisemblablement coûteux

Transactions
Concurrence d'accès

- Plusieurs transactions accèdent simultanément aux données
- Premier problème : lecture impropre

Transaction 1	**Transaction** 2
V=100	
	Lire V
ROLLBACK	

- Cette anomalie est corrigée par le définition des transactions : les valeurs modifiées ne seront visibles qu'après la fin de la transaction « verrous » (impact majeur sur les performances)

Transactions
Concurrence d'accès

- Deuxième problème : lecture non reproductible

Transaction 1	**Transaction** 2
`Lire V`	
	`V = V + 100`
	`COMMIT`
`Lire V`	

- Lourd à éviter pour les lectures
- Pour les modifications envisagées, nécessiter de verrouiller explicitement les données: `SELECT ... FOR UPDATE`

Transactions
Concurrence d'accès (suite)

- Troisième problème : lignes fantômes

Transaction 1	**Transaction** 2
SELECT…	
WHERE C=10	
	UPDATE…
	SET C=20
SELECT…	
WHERE C=10	

Transactions
Concurrence d'accès (suite)

- Quels sont les mécanismes misent en place par les sgbd pour gérer les accès concurrents?

- Premier mécanisme : la lecture consistante (*statement-level read consistency*)
- Second mécanisme : isolation et sérialisabilité : traduisent les interactions entre transaction

Transactions
Concurrence d'accès (suite)

- Premier mécanisme : la lecture consistante (*statement-level read consistency*) : les résultats d'une requête reflètent l'état des données au début de la requête
  - Pertinent dans le cas de requêtes longues : la fin est synchrone avec le début, même en cas de `COMMIT` par une autre transaction pendant le déroulement de la requête
  - Attention : concerne également `UPDATE` et `DELETE`, qui exécutent une requête.

Transactions
Concurrence d'accès (suite)

- Second mécanisme : isolation et sérialisabilité : traduisent les interactions entre transaction

- Par exemple, Oracle, PSQL, … fournit des niveaux d'isolation

Ex:

- 1 *Read committed* (par défaut) : lecture consistante, mais problèmes de lecture non-reproductible et de lignes fantômes

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

- 2 *Serializable* : la transaction ne voit que l'état au début de la transaction, plus ses propres modifications

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

- 3 *Read-only* : la transaction ne voit que l'état au début de la transaction, les modifications sont interdites

```
SET TRANSACTION READ ONLY
```

Transactions
Concurrence d'accès (suite)

- Le verrouillage (implicite ou explicite) consiste à associer à chaque élément de la base un « *verrou* » : libre ou occupé

- Avant chaque accès : test du verrou, et attente de libération si nécessaire
  - Verrou acquis par opération sur l'élément
  - Pas de libération de verrou en cours de transaction

- En général, 2 types de verrous :
  - `SHARE` : verrou partagé (« en lecture »), disponible pour plusieurs transaction simultanément
  - `EXCLUSIVE` : verrou exclusif (« en écriture »), une seule transaction à la fois

Transactions
Concurrence d'accès (suite)

- Les SGBD distinguent souvent deux niveaux de verrouillage : *ligne* et *table*
- Les opérations `INSERT, UPDATE, DELETE` placent en général un verrou exclusif sur les lignes concernées
- `SELECT ... FOR UPDATE` est un moyen standard de verrouiller des lignes pour modification éventuelle future
- Le verrouillage peut être explicite :

```
LOCK TABLE table IN {SHARE|EXCLUSIVE} MODE
   [NOWAIT]
```

Transactions
Concurrence d'accès (suite) - Interblocage

- Les problèmes habituels du verrouillage peuvent se poser : le *deadlock*

Transaction 1	**Transaction** 2
LOCK TABLE T1 IN EXCLUSIVE MODE	
	LOCK TABLE T2 IN EXCLUSIVE MODE
LOCK TABLE T2 IN EXCLUSIVE MODE	
	LOCK TABLE T1 IN EXCLUSIVE MODE

- En général, les SGBD empêchent l'apparition de *deadlocks*

Utilisateurs et privilèges

- Chaque utilisateur d'une base de données est identifié (accès avec mot de passe, sauf exception)
- Chaque objet d'un schéma a un propriétaire, lequel peut attribuer des privilèges à d'autres utilisateurs
- En général, l'administrateur a un pouvoir (presque) total
- Un utilisateur est crée par :

  ```
  CREATE USER nom IDENTIFIED BY mdp
  ```

  En général, accompagné de quantité de paramètres (quotas, etc.)
- On le supprime par :

  ```
  DROP USER
  ```

- Très souvent : possibilité d'« importer » les utilisateurs systèmes, ou d'un autre dispositif (annuaire, etc.)

Utilisateurs et privilèges
Privilèges

- Un privilège est un droit d'accès à certains objets
- SQL : attribution

```
GRANT priv [, priv]* ON object
TO user [WITH GRANT OPTION]
```

- Les privilèges incluent au moins :
  - ALTER
  - DELETE
  - EXECUTE
  - INDEX
  - INSERT
  - REFERENCES
  - SELECT
  - UPDATE

Sur tables et vues quand cela fait sens (sauf EXECUTE sur procédures)

Utilisateurs et privilèges
Privilèges (suite)

- Certains privilèges peuvent être restreints à certaines colonnes (`SELECT, UPDATE, INSERT, REFERENCES`)
- `WITH GRANT OPTION` attribue au bénéficiaire le droit de transmettre le privilège
- Révocation :

```
REVOKE priv [, priv]* ON object
FROM user [CASCADE CONSTRAINTS]
```

(supprime les contraintes définies au titre du privilège supprimé)

- La révocation s'applique également aux utilisateurs ayant reçu le privilège du révoqué

Utilisateurs et privilèges
Rôles

- Collection (nommée) de privilèges, attribués en bloc
- Création en deux temps :

```
CREATE ROLE nomrole
GRANT priv [, priv]* TO role
```

- Association d'un utilisateur

```
GRANT nomrole TO user
```

(idem pour `REVOKE`)

- Rôles souvent mis en évidence par l'analyse du système
-> politique de sécurité (globale)

Fin du TP 3

Cinquème Partie

PL / SQL