Cours de bases de données, aspects systèmes, http://sys.bdpedia.fr

Niveaux d'isolation

Philippe Rigaux

le cnam

Ce qu'on doit savoir

Il existe plusieurs niveaux d'isolation, du plus permissif au plus strict.

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.

Le niveau d'isolation par défaut n'est jamais le plus strict. Car

- inutile dans la grande majorité des cas ;
- provoque rejets et blocages difficiles à expliquer à l'utilisateur.

Quand l'isolation totale est nécessaire, il faut l'indiquer explicitement.

Niveaux d'isolation SQL

Définis en fonction des anomalies : lectures sales, lectures non répétables, tuples fantômes.

- `read uncommitted` : tout est permis, toutes les anomalies sont possibles.
- `read committed` : lectures sales non permises.
- `repeatable read` : seuls les tuples fantômes sont permis.
- `serializable` : isolation totale, aucune anomalie, interblocages possibles.

Niveau par défaut : `read committed` (Oracle) ou `repeatable read` (MySQL, PostgreSQL).

Il faut se mettre en mode `serializable` pour les processus transactionnels.

Le mode `read committed`

Retenir : une requête accède à l'état de la base **au moment où la requête est exécutée**.

Pas de lecture sale, car donnée en cours de modification ne fait pas partie de l'état de la base.

Assez fluide, mais autorise beaucoup d'anomalies (mises à jour perdues, contrôle).

Démonstration : on peut lire deux fois le même tuple dans une transaction et obtenir des résultats différents.

Le mode `repeatable read`

Retenir : 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.

Démonstration : on peut lire n fois le même tuple dans une transaction et obtenir toujours le même résultat.

Est-ce suffisant ?

Non, car des exécutions non sérialisables restent quand même possibles.

Démonstration : On prend notre base et on déroule l'exécution des mises à jour perdues.

Le mode `serializable`

Retenir : garantit l'isolation totale, et donc la cohérence de la base.

mais ... : risque non négligeable de **rejet** de l'une des transactions.

On se place dans ce mode avec la commande suivante, au début du code de la transaction.

```
set transaction isolation level serializable;
```

Démonstration...

La clause `for update`

Fait : le problème vient du fait qu'on **lit** une donnée pour la **modifier** ensuite.

- Le système ne peut pas le deviner !
- Il est obligé de tracer toutes les lectures (coût élevé)
- Il pose des verrous faibles $\rightarrow$ rejet quand les choses tournent mal

La clause `for update`

- Le programmeur déclare qu'une lecture va être suivie d'une mise à jour
- Le système pose un verrou fort pour celles-là, pas de verrou pour les autres

Permet de monter le niveau de blocage uniquement quand c'est nécessaire ... mais repose sur le facteur humain, pas assez fiable.

À retenir

Savoir repérer les transactions dans une application. Elle doivent respecter la **cohérence applicative** (le système ne peut pas la deviner pour vous).

Dans ce cas, appliquer le mode sérialisable.

La clause `for update` est en théorie meilleure, mais repose sur le facteur humain : probablement à éviter.

Savoir qu'en mode sérialisable on risque des rejets.

Comprendre les risques dans les autres modes.