

Isolation des transactions, oui, mais à quel niveau ?

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 28 Juillet 2010

<http://www.bortzmeyer.org/transactions-isolees.html>

Un des services les plus importants que rendent les SGBD est d'**isoler** les transactions les unes des autres, c'est-à-dire de faire en sorte que, pendant qu'une transaction joue avec les données, les autres pourront continuer à travailler tranquillement, avec des données stables. Mais certains débutants en SGBD ignorent qu'il existe plusieurs niveaux d'isolation dans SQL et que tous ne garantissent pas une isolation parfaite, loin de là.

Prenons l'exemple de PostgreSQL, qui documente très bien ces différents niveaux <<http://www.postgresql.org/docs/current/interactive/mvcc.html>> d'isolation. On va créer une table pour essayer :

```
essais=> CREATE TABLE Foo (id SERIAL, name TEXT, number INTEGER);
```

Maintenant, si on se connecte et qu'on lance une transaction, par défaut, celle-ci aura le niveau d'isolation `read committed`. Cela vaut dire qu'on verra les `COMMIT` des autres (ce que la littérature SQL appelle un "*phantom read*") :

```
essais=> BEGIN;
BEGIN
essais=> SELECT * FROM foo;
 id | name | number
----+-----+-----
(0 rows)

[Ici, une autre transaction fait un
"INSERT INTO Foo (name, number) VALUES ('durand', 15);"]

essais=> SELECT * FROM foo;
 id | name | number
----+-----+-----
  1 | durand |      15
(1 rows)
```

On n'a donc pas été complètement isolé des autres transactions. Si on veut le faire, il faut indiquer explicitement, après le début de la transaction mais avant la première requête, un niveau plus isolant :

```
essais=> BEGIN;
BEGIN
essais=> SET TRANSACTION isolation level serializable;
SET
essais=> SELECT * FROM foo;
  id | name | number
-----+-----
   1 | durand |    15
(1 rows)

[Cette fois, même si une autre transaction fait, par exemple, "INSERT
INTO Foo (name, number) VALUES ('dupont', 1)", nous ne le verrons pas
tant que notre propre transaction est en cours.]

essais=> SELECT * FROM foo;
  id | name | number
-----+-----
   1 | durand |    15
(1 rows)

[Lorsque notre transaction se termine, on voit les changements faits
entre temps.]

essais=> COMMIT;
COMMIT
essais=> SELECT * FROM foo;
  id | name | number
-----+-----
   1 | durand |    15
   2 | dupont |     1
(2 rows)
```

Pourquoi tout le monde n'utilise pas ce niveau maximal d'isolation ? Pourquoi n'est-il pas la valeur par défaut ? Parce qu'isoler a un coût, oblige le SGBD à plus de travail et à utiliser davantage de verrous. Pire, cela peut conduire à des cas où on a une erreur dans l'une des transactions. Prenons le cas où, pendant qu'on lit gentiment la table, une autre transaction la modifie avec UPDATE Foo SET number=6 WHERE id=2; . Par défaut, on verra ce changement dans notre propre transaction. Mais si on cherche la bagarre et qu'on demande à être complètement isolé :

```
essais=> BEGIN;
BEGIN
essais=> SET transaction ISOLATION LEVEL serializable;
SET
essais=> SELECT * FROM foo;
...
  1 | durand |    15
...

[Ici, une autre transaction modifie cette ligne. On ne voit pas le
changement, comme prévu.]

essais=> SELECT * FROM foo;
...
  1 | durand |    15
...

[Mais si on demande à changer cette ligne ?]

essais=> DELETE FROM foo WHERE id=1;
ERROR:  could not serialize access due to concurrent update
```

Cette fois, on a eu un **conflit**. PostgreSQL ne pouvait pas faire les deux opérations demandées. L'une d'elles se termine donc en erreur.

Si l'autre transaction ne faisait pas de `COMMIT` immédiatement, PostgreSQL est dans l'incertitude, il ne sait pas encore laquelle faire échouer car il ne sait pas s'il n'y aura pas plutôt un `ROLLBACK` :

```
essais=> BEGIN;
BEGIN
essais=> SET transaction ISOLATION LEVEL serializable;
SET
essais=> SELECT * FROM foo WHERE id=6;
 id | name | number
-----+-----+-----
   6 | armand |      8
(1 row)
```

[Ici, une autre transaction modifie cette ligne. On ne voit pas le changement, comme prévu. Mais :]

```
essais=> DELETE FROM foo WHERE id=6;
```

[Ici, la transaction est bloquée. `DELETE` ne rendra pas la main avant que l'autre transaction n'aie décidé d'un `COMMIT` ou d'un `ROLLBACK`. On voit que cela peut être gênant pour certaines applications.]

Si l'autre transaction fait un `COMMIT`, on est ramené au cas précédent : le `DELETE` se termine en erreur. Si l'autre transaction fait un `ROLLBACK` (renonce à son opération de mise à jour), alors le `DELETE` peut se terminer normalement.

Attention pour le comportement par défaut si on utilise une bibliothèque d'accès à PostgreSQL. Celle-ci peut mettre automatiquement dans un niveau d'isolation différent. Pour Python, la norme d'interface avec les SGBD [<http://www.python.org/dev/peps/pep-0249/>](http://www.python.org/dev/peps/pep-0249/) n'en parle pas. Avec `psycopg` [<http://initd.org/psycopg/>](http://initd.org/psycopg/), ce n'est pas le cas, le niveau par défaut [<http://initd.org/psycopg/docs/extensions.html#isolation-level-constants>](http://initd.org/psycopg/docs/extensions.html#isolation-level-constants) est le `read committed`. Ce petit programme Python (en ligne sur <http://www.bortzmeyer.org/files/isolation.py>) permet de tester. Il ouvre une transaction et affiche régulièrement la valeur d'une des lignes. Par défaut, il montre les changements que fait une autre transaction pendant ce temps.

PostgreSQL, pour réaliser cette isolation, utilise un mécanisme nommé MVCC. Son principal slogan est que « les écritures ne bloquent jamais les lectures et les lectures ne bloquent jamais les écritures ». (Dans l'exemple plus haut où `DELETE` était bloqué, c'est parce que l'autre opération était également une écriture, un `UPDATE`.)

Les essais ci-dessus étaient faits avec PostgreSQL. Mais cela serait à peu près pareil avec tous les SGBD sérieux. Par exemple, pour Oracle, on peut lire « *On Transaction Isolation Levels* » [<http://www.oracle.com/technology/oramag/oracle/05-nov/o65asktom.html>](http://www.oracle.com/technology/oramag/oracle/05-nov/o65asktom.html) » ou « *How Serializable Transactions Interact* » [<http://download.oracle.com/docs/cd/B12037_01/appdev.101/b10795/adfns_sq.htm#1020206>](http://download.oracle.com/docs/cd/B12037_01/appdev.101/b10795/adfns_sq.htm#1020206) ». Pour les autres modèles de SGBD, voir la bonne liste de Wikipédia.

Ceux et celles qui veulent lire davantage peuvent regarder un bon cours en français [<http://www2.lifl.fr/~durif/bdd/coursBD/html/transactionIsolation.html>](http://www2.lifl.fr/~durif/bdd/coursBD/html/transactionIsolation.html) ou bien une bonne discussion [<http://stackoverflow.com/questions/997728/which-isolation-mode-should-you-choose-i>](http://stackoverflow.com/questions/997728/which-isolation-mode-should-you-choose-i) sur Stack Overflow [<http://www.bortzmeyer.org/stack-overflow.html>](http://www.bortzmeyer.org/stack-overflow.html), qui explique bien pourquoi un haut niveau d'isolation n'est pas forcément souhaitable (il diminue le parallélisme).