SELECT FOR UPDATE en SQL, pour les accès concurrents

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 17 octobre 2007. Dernière mise à jour le 14 février 2019

https://www.bortzmeyer.org/select-for-update.html

Un problème courant avec les bases de données est l'accès à une liste de tâches, stockée dans la base, et à laquelle plusieurs programmes clients accèdent. Comment faire pour éviter qu'ils choisissent la même tâche ?

Si on veut que chaque tâche soit traitée une fois et une seule, et dans l'ordre où elles ont été enregistrées dans la base, la méthode typique est d'utiliser la requête SQL SELECT FOR UPDATE. Voyons d'abord ce qui se passe si aucune précaution particulière n'est prise.

Nous avons une liste de tâches dans la base, ainsi décrite :

```
CREATE TABLE tasks (id SERIAL UNIQUE,
  todo TEXT,
  done BOOLEAN DEFAULT false,
  owner INTEGER); -- The one who act on the task
```

Supposons maintenant que deux programmes clients, que nous nommerons Tancrède et Clorinde, accèdent à la base en même temps. Chacun va chercher la tâche la plus ancienne (de `id` minimale) non encore faite, puis la faire (et mettre à jour les champs `id` et `owner`).

Les essais sont faits avec PostgreSQL et son programme psql (attention, d'autres SGBD peuvent avoir des comportements par défaut différents). On lance deux terminaux et, dans chacun d'eux, on se connecte à la même base avec psql. Cela permet de voir l'effet de la concurrence entre ces deux applications clientes. (J'ai triché un peu en modifiant l'invite de psql pour afficher le nom de l'utilisateur) :

```
tancrède=> BEGIN;
BEGIN
tancrède=> SELECT min(id) FROM tasks WHERE NOT done;
 min
-----
   1
(1 row)

tancrède=> UPDATE tasks SET done=true,owner=1 WHERE id = 1;
UPDATE 1
tancrède=> COMMIT;
COMMIT
tancrède=> SELECT * FROM tasks;
 id |     todo     | done |   owner
----+--------------+------+-------------
  2 | Nothing more | f    |
  3 | Later        | f    |
  1 | Nothing      | t    |           1
(3 rows)
```

Clorinde verra exactement la même chose. La transaction lancée par le BEGIN fait qu'elle ne voit pas les modifications de Tancrède, elle récupère le même `id` et va accomplir la même tâche. Les modifications gagnantes seront simplement celles du dernier à avoir commité.

Par défaut, PostgreSQL a des transactions en isolation "*READ COMMITTED*". On peut augmenter leur niveau d'isolation :

```
tancrède=> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

qui permettra de détecter le problème mais pas de le résoudre. La deuxième transaction à faire le UPDATE sera simplement rejetée. C'est plus satisfaisant mais pas très pratique.

Une autre approche serait d'utiliser des verrous explicites. Cette solution n'est pas tellement dans l'esprit des bases de données à transactions et peut se payer cher en terme de performance. (La documentation de PostgreSQL explique ces différents choix <`https://www.postgresql.org/docs/current/interactive/mvcc.html`>.)

Une meilleure approche est le SELECT FOR UPDATE. Avec cette option, le SELECT va verrouiller automatiquement les données.

```
tancrède=> SELECT id FROM tasks WHERE NOT done ORDER BY id FOR UPDATE OF tasks;
```

On note qu'on ne peut plus utiliser la fonction `min()`, PostgreSQL n'autorisant pas les fonctions agrégat pour le SELECT FOR UPDATE. Il faudra donc récupérer plusieurs tâches et ne garder que la plus ancienne.

Si Clorinde tente un SELECT FOR UPDATE un peu après, son SELECT sera bloqué jusqu'à la fin de la transaction de Tancrède.

Ce mécanisme est bien expliqué dans la documentation de PostgreSQL <`http://www.postgresql.org/docs/current/interactive/applevel-consistency.html`>.

Si l'idée de rester bloqué lors d'un SELECT FOR UPDATE est désagréable, notons qu'il existe une option NOWAIT qu'on peut ajouter à la fin de la requête SQL. Son effet sera de renvoyer immédiatement une erreur si le SELECT FOR UPDATE est bloqué par un autre.

Notons enfin un dernier piège (et merci à Tom Lane pour ses explications). Je n'ai pas utilisé LIMIT 1 dans les SELECT ci-dessus alors que cela aurait été un moyen simple de ne récupérer qu'une seule tâche. C'est parce que LIMIT est évalué avant le FOR UPDATE. Un SELECT avec LIMIT 1 peut donc ne rien renvoyer du tout. L'application cliente qui veut quand même l'utiliser doit donc se préparer à ne rien recevoir et à reessayer ensuite de temps en temps.

MariaDB a un comportement par défaut identique (attention, cela dépend de beaucoup de choses, comme le type de base de données utilisée, car MariaDB peut en utiliser plusieurs). L'isolation par défaut est "*REPEATABLE READ*" <https://mariadb.com/kb/en/library/set-transaction/#repeatable-read> et le second client reçoit la même tâche, puis est bloqué lors de l'UPDATE puis décoincé lors du COMMIT du premier, écrasant ses modifications. Comme pour PostgreSQL, le FOR UPDATE <https://mariadb.com/kb/en/library/for-update/> permet de ne pas récupérer la même tâche.

Merci à Frédéric Brouard pour ses précisions.