A. Verrous

1. Concept des verrous

a. Un exemple de la vie courante

Imaginez qu'un fichier soit partagé sur un réseau d'entreprise par les membres d'un projet. Ce fichier permet à chacun d'indiquer les tâches qui ont été terminées. Tant qu'un seul membre de l'équipe accède à un moment donné à ce fichier, tout se passe bien : cette personne peut lire ou modifier les informations contenues dans le fichier, enregistrer les modifications le cas échéant et fermer le fichier.

La situation se complique si, par exemple, deux personnes ouvrent le fichier au même moment pour en changer le contenu. Si aucune précaution n'est prise, les modifications faites par les deux utilisateurs risquent de se chevaucher et de rendre le document illisible. Il est même possible qu'une modification de l'un des utilisateurs ne soit pas enregistrée du tout.

Les accès concurrents sur ce fichier peuvent donc conduire à des corruptions ou des pertes de données. Comment empêcher ces problèmes ? Une solution simple consiste à verrouiller le fichier dès qu'un utilisateur l'ouvre, de manière à ce qu'un autre utilisateur tentant d'ouvrir le fichier simultanément se trouve contraint d'attendre que le premier utilisateur ait terminé ses actions.

Vous pouvez améliorer cette stratégie si vous savez quel type d'action un utilisateur s'apprête à effectuer : lecture ou écriture. En effet, il n'est pas gênant que plusieurs utilisateurs lisent simultanément le fichier, il faut simplement empêcher qu'un utilisateur fasse des modifications pendant ce temps. Par contre, dans le cas d'une modification, vous devrez absolument vous assurer que le fichier n'est pas en cours d'utilisation.

Une conséquence importante d'un tel mécanisme de verrouillage est la sérialisation des accès, pouvant provoquer des attentes chez les utilisateurs. En effet, si plusieurs utilisateurs peuvent lire la même ressource simultanément, les écritures doivent dans tous les cas se dérouler les unes après les autres.

b. Transposition à MySQL

Les accès concurrents sont la règle pour MySQL car le serveur est multithreadé, c'est-à-dire qu'il peut répondre à plusieurs clients simultanément. Des stratégies pour éviter les corruptions de données sont donc nécessaires.

En reprenant l'exemple du paragraphe précédent, si vous imaginez que le fichier est une table et les données une ligne d'une table, vous voyez que le mécanisme de verrouillage évoqué peut très bien s'appliquer pour gérer les accès concurrents à une base de données.

Avec MySQL, vous pouvez poser un verrou permettant à plusieurs clients d'accéder simultanément en lecture à la ressource verrouillée : on parle alors de verrou en lecture ou de verrou partagé. Aucun client ne peut accéder à la ressource pour la modifier, pas même celui qui a posé le verrou partagé.

Vous pouvez également poser un verrou donnant l'accès exclusif à la ressource verrouillée : on parle cette fois de verrou en écriture ou verrou exclusif. Dans ce cas, seul le client ayant posé le verrou peut accéder à la ressource, que ce soit en lecture ou en écriture.


InnoDB, avec son système de multiversionnement, permet d'accéder en lecture à des lignes protégées par un verrou exclusif.

Lorsqu'un verrou ne peut pas être acquis immédiatement, MySQL place la demande dans une file d'attente et fait attendre le client tant que sa demande ne peut pas être satisfaite.

Sachez que seul le client ayant posé le verrou a la possibilité de le lever et il n'est pas possible de lever un verrou posé par un autre client. En cas de déconnexion du client, le serveur lève automatiquement tous les verrous posés par le client durant sa session. Et en cas d'arrêt du serveur, l'ensemble des verrous sont relâchés.

En général, le serveur gère lui-même implicitement de manière interne la pose et la levée des verrous : le mécanisme reste transparent pour les utilisateurs. Mais en cas de besoin, l'utilisateur pourra poser explicitement des verrous.

c. Granularité des verrous

En fonction du moteur de stockage, le verrou, qu'il soit partagé ou exclusif, est posé sur la table dans son intégralité ou sur les seules lignes concernées par la requête. Ces deux granularités différentes ont des avantages et des inconvénients.

Les verrous sur les tables sont les plus simples à mettre en place par le serveur. Ils ont également l'avantage de ne pas permettre les interblocages (deadlocks) car MySQL acquiert tous les verrous nécessaires en début de requête. Mais ils ne permettent pas toujours d'obtenir de bonnes performances en cas de nombreux accès concurrents à la fois en lecture et en écriture : si par exemple, un client A veut

modifier une ligne d'une table contenant un million de lignes, A va bloquer l'accès à la table pour tous les autres clients qui voudraient lire dans la table, même ceux qui n'ont pas besoin d'accéder à la ligne modifiée.

Au contraire, les verrous sur les lignes demandent un surplus de travail au serveur et autorisent les interblocages. En contrepartie, les accès concurrents sont souvent plus performants : plusieurs clients peuvent par exemple écrire simultanément sur des lignes différentes de la même table, ou des clients peuvent lire des lignes sur la table alors que d'autres lignes sont modifiées au même instant.

Selon les cas, l'une ou l'autre méthode de verrouillage est meilleure, mais le choix entre les méthodes se limite à un choix du moteur de stockage : parmi les moteurs courants, `MyISAM` et `MEMORY` utilisent un verrouillage sur les tables et `InnoDB` un verrouillage sur les lignes.

La notion d'interblocage sera expliquée dans la suite de ce chapitre, dans la partie concernant les transactions.

2. Verrous implicites

a. Niveau de gestion des verrous implicites

Les requêtes de lectures et d'écritures (`SELECT`, `INSERT`, `UPDATE`...) posent un verrou implicite dont la granularité (table ou ligne essentiellement) est gérée par le moteur de stockage.

En cas de forts accès concurrents, les mêmes requêtes exécutées sur des tables contenant des données identiques mais n'ayant pas les mêmes moteurs peuvent avoir des performances complètement différentes, simplement à cause de la granularité du verrou.

Les commandes comme `REPAIR TABLE`, `OPTIMIZE TABLE` (voir le chapitre Maintenance des tables) ou `ALTER TABLE` provoquent également un verrouillage implicite des tables. Dans ce cas, le verrou est géré par le serveur lui-même et se fait toujours sur l'intégralité de la table, indépendamment du moteur de stockage et pour toute la durée de l'opération.

b. Spécificités MyISAM

Le moteur MyISAM utilise toujours des verrous sur les tables.

Quand des requêtes sont reçues pour des tables MyISAM, le serveur applique par défaut deux règles pour savoir dans quel ordre exécuter les requêtes :

- Les écritures (`INSERT`, `UPDATE`, `DELETE`, `REPLACE`) sont prioritaires par rapport aux lectures (`SELECT`).
- Les écritures sont exécutées dans l'ordre dans lequel elles sont reçues par le serveur (dans la mesure du possible).

Cela signifie que si un client attend qu'une table ne soit plus verrouillée pour lire des données et que d'autres clients veulent modifier la même table, ces autres clients vont pouvoir accéder à la table avant le premier client même si leur requête est arrivée après. À l'extrême, sur une table constamment sollicitée en écriture, une requête en lecture va attendre indéfiniment que les verrous soient levés.

MyISAM permet de modifier cet ordre de priorité avec des mots-clés spéciaux à placer dans les requêtes concernées :

- `DELAYED` peut s'utiliser avec `INSERT` et `REPLACE`. Le serveur n'exécute alors pas la requête immédiatement mais la place dans un tampon mémoire et redonne immédiatement la main au client. Périodiquement, le serveur va vérifier si les tables concernées sont en cours d'utilisation et dans le cas contraire, il va regrouper les instructions `DELAYED` et les exécuter.

Exemple d'utilisation de `DELAYED` :

```
mysql> INSERT DELAYED INTO log (user_id,action) VALUES
(15,'login');
```

Vous pourrez utiliser ce modificateur de priorité pour les insertions pouvant être différées dans le temps, comme par exemple les insertions dans des tables de log. Les verrous posés par ces requêtes non prioritaires seront alors réduits au minimum. De plus, MySQL réordonne ces instructions pour les exécuter dans l'ordre optimal. Mais attention, en cas de crash du serveur, des données risquent d'être perdues car les instructions n'existent qu'en mémoire.

- `LOW_PRIORITY` peut s'utiliser pour les requêtes opérant une écriture (`INSERT`, `UPDATE`, `REPLACE`, `DELETE`). Dans ce cas, le serveur attendra non seulement que tous les éventuels verrous posés sur la table par d'autres requêtes en cours d'exécution soient levés, mais également que toutes les requêtes en lecture

reçues par le serveur après la requête en écriture soient exécutées. Il s'agit d'un renversement des priorités par rapport à ce qui se produit par défaut.

Exemple d'utilisation pour une insertion dans une table de log :

```
mysql> INSERT LOW_PRIORITY INTO log (user_id,action) VALUES
(15,'login');
```

- `HIGH_PRIORITY` peut s'utiliser dans un `SELECT` pour faire en sorte que ce `SELECT` soit prioritaire sur toutes les requêtes en écriture en attente et sur tous les autres `SELECT` en attente et ne portant pas le mot-clé `HIGH_PRIORITY`.

Exemple d'utilisation pour une lecture dans une table de log :

```
mysql> SELECT HIGH_PRIORITY user_id FROM log WHERE action='login'
ORDER BY user_id LIMIT 100;
```

c. Spécificités InnoDB

Dans de nombreux cas, ce moteur de stockage n'a pas besoin de verrouiller les lignes pour les lectures. InnoDB dispose en effet d'un mécanisme de multiversionnement, appelé MVCC (*MultiVersion Concurrency Control*), qui permet de consulter un instantané des données et qui demande généralement moins de travail au serveur que le verrouillage de lignes.

Quand c'est nécessaire, InnoDB va avoir recours au verrouillage de lignes sans jamais escalader au verrouillage de toute la table. Dans certains cas cette stratégie est peu efficace, par exemple lorsqu'InnoDB va parcourir toutes les lignes de la table et donc les verrouiller les unes après les autres.


Des commandes comme `ALTER TABLE` posent un verrou sur toute la table même si celle-ci utilise InnoDB. Ce type de verrou est géré directement par le serveur et non pas par le moteur stockage.

InnoDB acquiert les verrous nécessaires au fur et à mesure qu'il estime que ces acquisitions sont nécessaires. Selon le plan d'exécution de la requête, cette façon de procéder peut conduire InnoDB à verrouiller beaucoup plus de lignes qu'on pourrait l'imaginer. Un exemple sera donné un peu plus loin dans ce chapitre.

L'association de lectures potentiellement sans verrou grâce au multiversionnement et de verrouillages au niveau des lignes rend InnoDB performant lorsque le serveur reçoit un mélange de requêtes en lecture et en écriture.

En résumé, avec InnoDB, il est possible que, simultanément sur la même table, plusieurs clients :

- lisent des lignes, identiques ou différentes ;
- lisent des lignes et écrivent d'autres lignes ;
- écrivent des lignes différentes.

Par contre, il n'est pas possible que plusieurs clients écrivent simultanément sur une même ligne d'une table.

3. Verrous explicites

a. Poser/enlever un verrou

MySQL offre la possibilité de verrouiller par une instruction une ou plusieurs tables. Le but n'est pas de remédier à des erreurs de verrouillage de la part du serveur, mais plutôt d'indiquer au serveur qu'on a besoin qu'une ou plusieurs tables soient verrouillées le temps d'effectuer certaines opérations.

D'une manière générale, un utilisateur peut demander au serveur de poser un verrou partagé qui permet à d'autres clients d'accéder en lecture seule aux tables verrouillées, ou un verrou exclusif qui empêche tout accès aux tables verrouillées. La commande à utiliser est `LOCK TABLES` avec plusieurs mots-clés possibles :

- `READ` pose un verrou partagé ;
- `WRITE` pose un verrou exclusif ;
- `READ_LOCAL` pose un verrou partagé permettant les insertions concurrentes pour les tables MyISAM. Cette fonctionnalité est détaillée au paragraphe suivant.

Exemple d'utilisation :

```
mysql> LOCK TABLES t1 READ, t2 READ, t3 WRITE;
```

Notez qu'il faut impérativement verrouiller en une seule instruction l'ensemble des tables que vous souhaitez verrouiller. En effet, l'instruction LOCK TABLES a pour effet implicite de déverrouiller toutes les tables qui auraient été précédemment verrouillées.