Transaction

Une transaction, c'est un ensemble de requêtes qui sont exécutées en un seul bloc. Ainsi, si une des requêtes du bloc échoue, on peut décider d'annuler tout le bloc de requêtes (ou de quand même valider les requêtes qui ont réussi). On peut donc considérer que chaque requête constitue une transaction qui est automatiquement commitée. Par défaut, MySQL est donc en mode "autocommit".

Pour quitter ce mode, il suffit de lancer la requête suivante :

```
set autocommit=0;
Query OK, 0 rows affected (0,00 sec)
```

Inserons une donnée dans une table

```
mysql> use transaction;
Database changed
mysql> create table testtransaction(i integer) engine='INNODB';
Query OK, 0 rows affected (0,09 sec)
```

Connection 1	Connection 2

`mysql> begin;`
`Query OK, 0 rows affected (0,00 sec)`
`mysql> insert into testtransaction values (2);`
`Query OK, 1 row affected (0,00 sec)`
`mysql> select * from testtransaction;`
	`mysql> select * from testtransaction;`
	
	`Empty set (0,01 sec)`

Lors du commit:

Connection 1	Connection 2

`mysql> commit ;`
`Query OK, 0 rows affected (0,10 sec)`
	`select * from testtransaction;`
	`+------+`
	`| i    |`
	`+------+`
	`|    2 |`
	`+------+`
	`1 row in set (0,00 sec)`

L'operation de rollback permet d'annuler une transaction:

```
mysql> begin;
Query OK, 0 rows affected (0,00 sec)

mysql> insert into testtransaction values (5);
Query OK, 1 row affected (0,00 sec)

mysql> select * from testtransaction;
+------+
| i    |
+------+
|    2 |
|    5 |
+------+
2 rows in set (0,00 sec)

mysql> rollback;
Query OK, 0 rows affected (0,01 sec)

mysql> select * from testtransaction;
+------+
| i    |
+------+
|    2 |
+------+
1 row in set (0,00 sec)
```

Il faut faire attention que les comit/rollback en concerne pas les opération ddl (create table, alter ...)

Connection 1	Connection 2
mysql> begin -> ; Query OK, 0 rows affected (0,00 sec) mysql> alter table testtransaction add column testcolumn integer; Query OK, 0 rows affected (0,12 sec) Records: 0 Duplicates: 0 Warnings: 0	mysql> select * from testtransaction; +------+------------+ | i | testcolumn | +------+------------+ | 2 | NULL | +------+------------+ 1 row in set (0,00 sec)

Lors du rollback:

Connection 1	Connection 2
mysql> rollback ; Query OK, 0 rows affected (0,00 sec)	mysql> select * from testtransaction; +------+------------+ | i | testcolumn | +------+------------+ | 2 | NULL | +------+------------+ 1 row in set (0,00 sec)

Transaction Level

Il est possible de savoir le niveau des transaction via:

```
-- mysql> show variables like 'tx_isolation';
-- sous mysql 8 tx_isolation a été renommé en transation_isolation
mysql> show  variables like 'transaction_isolation';
+-----------------------+-----------------+
| Variable_name         | Value           |
+-----------------------+-----------------+
| transaction_isolation | REPEATABLE-READ |
+-----------------------+-----------------+
1 row in set (0,00 sec)
```

Repeatable Read

C'est le niveau d'isolation par défaut pour InnoDB. Des lectures cohérentes dans la même transaction lisent l'instantané établi par la première lecture. Cela signifie que si vous émettez plusieurs instructions SELECT simples (non verrouillables) dans la même transaction, ces instructions SELECT sont également cohérentes.

Connection 1	Connection 2
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; Query OK, 0 rows affected (0,00 sec) mysql> create table transactionlevel (i integer); Query OK, 0 rows affected (0,12 sec) mysql> begin -> ; Query OK, 0 rows affected (0,00 sec) mysql> insert into transactionlevel values (2); Query OK, 1 row affected (0,00 sec)	mysql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; Query OK, 0 rows affected (0,00 sec) mysql> begin; Query OK, 0 rows affected (0,00 sec) mysql> select * from transactionlevel; Empty set (0,00 sec)

Apres le commit de la première connexion, on obtient:

Connection 1	Connection 2
commit; Query OK, 0 rows affected (0,06 sec) mysql> select * from transactionlevel; +------+ | i | +------+ | 2 | +------+ 1 row in set (0,00 sec)	mysql> select * from transactionlevel; Empty set (0,00 sec)

Il faut attendre le commit de la transaction pour voir les changements;

```
mysql> select * from transactionlevel;
Empty set (0,00 sec)

mysql> commit;
Query OK, 0 rows affected (0,00 sec)

mysql> select * from transactionlevel;
+------+
| i    |
+------+
|    2 |
+------+
1 row in set (0,00 sec)
```

Table Lock vs Row Lock

Avantages du verrouillage au niveau de la ligne:

- Moins de conflits de verrous lors de l'accès à différentes lignes dans de nombreux threads.
- Moins de changements pour les annulations.
- Permet de verrouiller une seule ligne pendant une longue période.

Inconvénients du verrouillage au niveau de la ligne:

- Prend plus de mémoire que les verrous au niveau de la page ou de la table.
- Est plus lent que les verrous au niveau de la page ou de la table lorsqu'il est utilisé sur une grande partie de la table car vous devez acquérir beaucoup plus de verrous.

Avec les verrous de niveau supérieur, vous pouvez également prendre en charge plus facilement des verrous de différents types pour optimiser l'application, car la charge de verrouillage est inférieure à celle des verrous de niveau ligne.

Les verrous de table sont supérieurs aux verrous de page ou de ligne dans les cas suivants:

- La plupart des instructions pour la table sont des lectures.

Table Lock

Read Lock

Un verrou READ présente les caractéristiques suivantes:

- Un verrou READ pour une table peut être acquis par plusieurs sessions simultanément. De plus, d'autres sessions peuvent lire des données de la table sans acquérir le verrou.
- La session qui détient le verrou READ peut uniquement lire les données de la table, mais ne peut pas écrire. De plus, les autres sessions ne peuvent pas écrire de données dans la table tant que le verrou READ n'est pas libéré. Les opérations d'écriture d'une autre session seront mises dans les états en attente jusqu'à ce que le verrou READ soit libéré.
- Si la session est terminée, normalement ou de manière anormale, MySQL libérera tous les verrous implicitement. Cette fonctionnalité est également pertinente pour le verrou WRITE.

Connection 1	Connection 2

`mysql> create table t (i integer);`
`Query OK, 0 rows affected (0.08 sec)`
 	
`mysql> lock table t read;`
`Query OK, 0 rows affected (0.00 sec)`
 	
`mysql> insert into t values(3);`
`ERROR 1099 (HY000): Table 't' was locked`
`with a READ lock and can't be updated`
	`mysql> insert into t values(3);`
	`-- la connexion est bloqué`

Write Lock

Un verrou WRITE présente les caractéristiques suivantes:

- La seule session qui détient le verrou d'une table peut lire et écrire des données à partir de la table.
- D'autres sessions ne peuvent pas lire et écrire des données dans la table tant que le verrou WRITE n'est pas libéré.

Connection 1	Connection 2

`mysql> lock table testtransaction write;`
`Query OK, 0 rows affected (0,00 sec)`
 	
`mysql> insert into testtransaction values(1,2);`
`Query OK, 1 row affected (0,00 sec)`
	`select * from testtransaction;`
	`-- la connection 2 est bloqué`

Statut des lock

Il est possible d'avoir le statut des connection via SHOW PROCESS LIST

```
*************************** 4. row ***************************
     Id: 10
   User: root
   Host: localhost:33736
     db: transaction
Command: Query
   Time: 28
  State: Waiting for table metadata lock
   Info: select * from testtransaction
4 rows in set (0,00 sec)
```

Kill

Il est parfois nécéssaire de tuer une transaction afin de libérer les lock.
Dans ce cas, on doit récuperer la liste des process et executer une requete KILL

```
SHOW PROCESSLIST \G;
*************************** 1. row ***************************
```

```
     Id: 4
   User: event_scheduler
   Host: localhost
     db: NULL
Command: Daemon
   Time: 16478
  State: Waiting on empty queue
   Info: NULL
*************************** 2. row ***************************
     Id: 8
   User: root
   Host: localhost:33716
     db: myisamtest
Command: Sleep
   Time: 5193
  State:
   Info: NULL
*************************** 3. row ***************************
     Id: 9
   User: root
   Host: localhost:33734
     db: transaction
Command: Sleep
   Time: 5
  State:
   Info: NULL
*************************** 4. row ***************************
     Id: 11
   User: root
   Host: localhost:33738
     db: transaction
Command: Query
   Time: 0
  State: starting
   Info: SHOW PROCESSLIST
4 rows in set (0,00 sec)

ERROR:
No query specified
```

```
mysql> kill 9
    -> ;
Query OK, 0 rows affected (0,00 sec)
```

KILL autorise un modificateur CONNECTION ou QUERY:

- KILL CONNECTION est identique à KILL sans modificateur: il met fin à la connexion associée à processlist_id donné, après avoir terminé toute instruction en cours d'exécution par la connexion.
- KILL QUERY met fin à l'instruction en cours d'exécution de la connexion, mais la laisse elle-même intacte.

Si vous avez le privilège PROCESSUS, vous pouvez voir tous les threads. Si vous disposez du privilège CONNECTION_ADMIN ou SUPER, vous pouvez supprimer tous les threads et toutes les instructions. Sinon, vous ne pouvez voir et tuer que vos propres discussions et déclarations.

Row Lock

Si vous interrogez des données puis insérez ou mettez à jour des données associées dans la même transaction, l'instruction SELECT standard ne donne pas une protection suffisante. D'autres transactions peuvent mettre à jour ou supprimer les mêmes lignes que celles que vous venez de consulter. InnoDB prend en charge deux types de lectures de verrouillage offrant une sécurité supplémentaire:

SELECT... FOR SHARE

Définit un verrou de mode partagé sur toutes les lignes lues. D'autres sessions peuvent lire les lignes, mais ne peuvent pas les modifier jusqu'à ce que votre transaction soit validée. Si l'une de ces lignes a été modifiée par une autre transaction qui n'a pas encore été validée, votre requête attend jusqu'à la fin de cette transaction, puis utilise les dernières valeurs.

Connection 1	Connection 2

```
mysql> begin ;
Query OK, 0 rows affected (0,00 sec)

mysql> select * from testtransaction where
i=2 FOR SHARE;
+------+------------+
| i    | testcolumn |
+------+------------+
|    2 |       NULL |
+------+------------+
1 row in set (0,00 sec)



mysql> insert into testtransaction
values(1,2);
Query OK, 1 row affected (0,00 sec)
```

```
 mysql> begin;
Query OK, 0 rows affected (0,02 sec)

mysql> update testtransaction set
testcolumn='pilou';
-- la transaction est bloqué
ERROR 1205 (HY000): Lock wait timeout
exceeded;
try restarting transaction
```

seule les lignes impacté par le select sont locké

Connection 1	Connection 2

```
mysql> select * from testtransaction where
i=2 FOR SHARE;
+------+------------+
| i    | testcolumn |
+------+------------+
|    2 |       NULL |
+------+------------+
1 row in set (0,00 sec)
```

```
mysql> begin;
Query OK, 0 rows affected (0,02 sec)

mysql> update testtransaction set
testcolumn=45 where i=1;
Query OK, 2 rows affected (0,00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

SELECT ... FOR UPDATE

Une commande SELECT ... FOR UPDATE va lire les dernières données disponibles pour chaque ligne, et pose un verrou dessus en même tant qu'il lit. De cette façon, il pose le même verrou que la commande UPDATE .

Connection 1	Connection 2

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from SolrCoresPreallocated
order by id limit 1 for update;
+----+-------------+-----+-----+
| id | used_status | sid | cid |
+----+-------------+-----+-----+
|  1 |           0 |   0 | 400 |
+----+-------------+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from SolrCoresPreallocated
order by id limit 1 for update;
ERROR 1205 (HY000): Lock wait timeout
exceeded; try restarting transaction
mysql>  rollback;
Query OK, 0 rows affected (0.00 sec)
```

NOWAIT

Si une ligne est verrouillée par une transaction, une transaction SELECT ... FOR UPDATE ou SELECT ... FOR SHARE demandant la même ligne verrouillée doit attendre que la transaction bloquante libère le verrou de ligne. Ce comportement empêche les transactions de mettre à jour ou de supprimer les lignes qui sont interrogées pour des mises à jour par d'autres transactions. Cependant, il n'est pas nécessaire d'attendre qu'un verrou de ligne soit libéré si vous souhaitez que la requête soit renvoyée immédiatement lorsqu'une ligne demandée est verrouillée, ou si le fait d'exclure des lignes verrouillées du jeu de résultats est acceptable.

Pour éviter d'attendre que d'autres transactions libèrent des verrous de ligne, les options NOWAIT et SKIP LOCKED peuvent être utilisées avec les instructions de lecture SELECT ... FOR UPDATE ou SELECT ... FOR SHARE.

NOWAIT

Une lecture de verrouillage qui utilise NOWAIT n’attend jamais d’acquérir un verrou de ligne. La requête s'exécute immédiatement, échouant avec une erreur si une ligne demandée est verrouillée.

SKIP LOCKED

Une lecture de verrouillage qui utilise SKIP LOCKED n’attend jamais d’obtenir un verrou de ligne. La requête s'exécute immédiatement, en supprimant les lignes verrouillées du jeu de résultats.

Revision #12
Created 24 November 2019 07:44:49 by Admin
Updated 26 November 2019 19:23:33 by Admin