15 全局锁和表锁什么场景会用到

更新时间：2019-09-03 10:33:11

> 既然我已经踏上这条道路，那么，任何东西都不应妨碍我沿着这条路走下去。
>
> ——康德

从这一节开始，我们进入专栏的第三个 MySQL 知识大类：MySQL 锁。

在上一章（MySQL 索引）中，我们介绍了索引原理、需要添加索引的场景、一些常见索引类型的区别等，以及分享了有些场景 MySQL 会选错索引及选错索引时的处理方式等。通过这些学习，我们知道了提高查询效率的方法。但是，数据库往往是多个用户或者客户端在连接使用的。这时，我们需要考虑一个新的问题：**如何保证数据并发访问的一致性、有效性呢？**

MySQL 中，锁就是协调多个用户或者客户端并发访问某一资源的机制，保证数据并发访问时的一致性和有效性。

本章就来介绍一下不同场景下的锁机制。

根据加锁的范围，MySQL 中的锁可分为三类：

- 全局锁
- 表级锁
- 行锁

本节来重点讲解一下全局锁和表锁。

1 全局锁

MySQL 全局锁会关闭所有打开的表，并使用全局读锁锁定所有表。其命令为：

```
FLUSH TABLES WITH READ LOCK;
```

简称：FTWRL，可以使用下面命令解锁：

```
UNLOCK TABLES;
```

我们来通过实验理解一下全局锁：

首先创建测试表，并写入数据：

```
use muke;
drop table if exists t14;
CREATE TABLE `t14` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `a` int(11) NOT NULL,
  `b` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `idx_a` (`a`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
insert into t14(a,b) values(1,1);
```

进行 FTWRL 实验：

session1	session2
FLUSH TABLES WITH READ LOCK; Query OK, 0 rows affected (0.00 sec)	
select * from t14 limit 1; … 1 row in set (0.00 sec) （能正常返回结果）	select * from t14 limit 1; … 1 row in set (0.00 sec) （能正常返回结果）
insert into t14(a,b) values(2,2); ERROR 1223 (HY000): Can't execute the query because you have a conflicting read lock （报错）	insert into t14(a,b) values(2,2);/* sql1 */ （等待）
UNLOCK TABLES;	insert into t14(a,b) values(2,2);/* sql1 */ Query OK, 1 row affected (5.73 sec) （**session1** 解锁后，在等待的 **sql1** 马上执行成功）

上面的实验中，当 session1 执行 FTWRL 后，本线程 session1 和其它线程 session2 都可以查询，本线程和其它线程都不能更新。

原因是：**当执行 FTWRL 后，所有的表都变成只读状态，数据更新或者字段更新将会被阻塞。**

那么全局锁一般什么时候会用到呢？

全局锁一般用在整个库（包含非事务引擎表）做备份（mysqldump 或者 xtrabackup）时。也就是说，在整个备份过程中，整个库都是只读的，其实这样风险挺大的。如果是在主库备份，会导致业务不能修改数据；而如果是在从库备份，就会导致主从延迟。

好在 mysqldump 包含一个参数 --single-transaction，可以在一个事务中创建一致性快照，然后进行所有表的备份。因此增加这个参数的情况下，备份期间可以进行数据修改。但是需要所有表都是事务引擎表。所以这也是建议使用 InnoDB 存储引擎的原因之一。

而对于 xtrabackup，可以分开备份 InnoDB 和 MyISAM，或者不执行 --master-data，可以避免使用全局锁。

2 表级锁

表级锁有两种：表锁和元数据锁。

2.1 表锁

表锁使用场景：

1. 事务需要更新某张大表的大部分或全部数据。如果使用默认的行锁，不仅事务执行效率低，而且可能造成其它事务长时间锁等待和锁冲突，这种情况下可以考虑使用表锁来提高事务执行速度；
2. 事务涉及多个表，比较复杂，可能会引起死锁，导致大量事务回滚，可以考虑表锁避免死锁。

其中表锁又分为表读锁和表写锁，命令分别是：

表读锁：

```
lock tables t14 read;
```

表写锁：

```
lock tables t14  write;
```

下面我们分别用实验验证表读锁和表写锁。

表读锁实验：

session1	session2
lock tables t14 read; Query OK, 0 rows affected (0.00 sec)	
select id,a,b from t14 limit 1; … 1 row in set (0.00 sec) （能正常返回结果）	select id,a,b from t14 limit 1; … 1 row in set (0.00 sec) （能正常返回结果）
insert into t14(a,b) values(3,3); ERROR 1099 (HY000): Table ‘t14’ was locked with a READ lock and can’t be updated （报错）	insert into t14(a,b) values(3,3);/* sql2 */ （等待）
unlock tables; Query OK, 0 rows affected (0.00 sec)	insert into t14(a,b) values(3,3);/* sql2 */ Query OK, 1 row affected (10.97 sec) （**session1** 解锁后，**sql2** 立马写入成功）

从上面的实验我们可以看出，在 session1 中对表 t14 加表读锁，session1 和 session2 都可以查询表 t14 的数据；而 session1 执行更新会报错，session2 执行更新会等待（直到 session1 解锁后才更新成功）。

总结：**对表执行 lock tables xxx read （表读锁）时，本线程和其它线程可以读，本线程写会报错，其它线程写会等待。**

我们再来看一下表写锁实验：

session1	session2
lock tables t14 write; Query OK, 0 rows affected (0.00 sec)	
select id,a,b from t14 limit 1; … 1 row in set (0.00 sec) （能正常返回结果）	select id,a,b from t14 limit 1;/* sql3 */ （等待）

session1	session2
unlock tables; Query OK, 0 rows affected (0.01 sec)	select id,a,b from t14 limit 1;/* sql3 */ … 1 row in set (7.16 sec) （**session1** 解锁后，**sql3** 马上返回查询结果）
lock tables t14 write; Query OK, 0 rows affected (0.00 sec)	
delete from t14 limit 1; Query OK, 1 row affected, 1 warning (0.00 sec) （能正常执行删除语句）	delete from t14 limit 1;/* sql4 */ （等待）
unlock tables; Query OK, 0 rows affected (0.00 sec)	delete from t14 limit 1;/* sql4 */ Query OK, 1 row affected, 1 warning (14.94 sec) （**session1** 解锁后，**sql4** 立马执行成功）

总结：对表执行 **lock tables xxx write** （表写锁）时，本线程可以读写，其它线程读写都会阻塞。

2.2 元数据锁

在 MySQL 中，DDL 是不属于事务范畴的。如果事务和 DDL 并行执行同一张表时，可能会出现事务特性被破坏、binlog 顺序错乱等 bug（比如 bug#989）。为了解决这类问题，从 MySQL 5.5.3 开始，引入了元数据锁（Metadata Locking，简称：MDL 锁）（这段内容参考《淘宝数据库内核月报》MySQL · 特性分析 · MDL 实现分析）。

从上面我们知道，MDL 锁的出现解决了同一张表上事务和 DDL 并行执行时可能导致数据不一致的问题。

但是，我们在工作中，很多情况需要考虑 MDL 的存在，否则可能导致长时间锁等待甚至连接被打满的情况。如下例：

session1	session2	session3
select id,a,b,sleep(100) from t14 limit 1;/* sql5 */		
	alter table t14 add column c int;/* sql6 */ （等待）	select id,a,b from t14 limit 1;/* sql7 */ （等待）
select id,a,b,sleep(100) from t14 limit 1;/* sql5 */ … 1 row in set (1 min 40.00 sec) （**100**秒后 **sql5** 返回结果）	alter table t14 add column c int;/* sql6 */ Query OK, 0 rows affected (1 min 33.98 sec) Records: 0 Duplicates: 0 Warnings: 0 （**session1** 的查询语句执行完成后，**sql6** 立马执行完毕）	select id,a,b from t14 limit 1;/* sql7 */ … 1 row in set (1 min 26.65 sec) （**session1** 的查询语句执行完成后，**sql7** 立马执行完毕）

上面的实验中，我们在 session1 查询了表 t14 的数据，其中使用了 sleep(100) ，表示在 100 秒后才会返回结果；然后在 session2 执行 DDL 操作时会等待（原因是 session1 执行期间会对表 t14 加一个 MDL，而 session2 又会跟 session1 争抢 MDL）；而 session3 执行查询时也会继续等待。因此如果 session1 的语句一直没结束，其它所有的查询都会等待。这种情况下，如果这张表查询比较频繁，很可能短时间把数据库的连接数打满，导致新的连接无法建立而报错，如果是正式业务，影响是非常恐怖的。

当然如果出现这种情况，假如你还有 session 连着数据库，可以 kill 掉 session1 中的语句或者终止 session2 中的 DDL 操作，可以让业务恢复。但是出现这种情况的根源其实是：session1 中有长时间未提交的事务。**因此对于开发来说，在工作中应该尽量避免慢查询、尽量保证事务及时提交、避免大事务等，当然对于 DBA 来说，也应该尽量避免在业务高峰执行 DDL 操作。**

3 总结

本节讲解了全局锁和表锁。

其中**全局锁会让所有的表变成只读状态，所有更新操作都会被阻塞。**

而表级锁分为表锁和元数据锁。

表锁又提到了表读锁和表写锁，并都进行了实验。两者的区别是：

表读锁：本线程和其它线程可以读，本线程写会报错，其它线程写会等待。

表写锁：本线程可以读写，其它线程读写都会阻塞。

为了保证事务和 DDl 并行执行数据一致，在 MySQL 5.5.3 引入了 MDL 锁。通过本节讲解的 MDL 锁机制，应该注意的几个点是：

- 尽量避免慢查询
- 事务要及时提交
- 避免大事务
- 避免在业务高峰执行 DDL 操作

4 问题

一张几百行数据的小表，在业务高峰执行 DDL， 是不是不会出现因为 MDL 而导致连接被打满的情况？

5 参考资料

《深入浅出 MySQL》第二版：20.3.8 什么时候使用表锁

《MySQL 5.7 参考手册》：13.7.6.3 FLUSH Syntax

《淘宝数据库内核月报》：MySQL · 特性分析 · MDL 实现分析

}

14 为什么MySQL会选错索引？

16 行锁：InnoDB替代MyISAM的重要原因