Java MySQL系列课程10

------ MVCC&表锁&行锁&间隙锁&死锁

本文档包含以下内容：

1. MyISAM表锁
2. InnoDB行锁
3. 间隙锁
4. InnoDB死锁

#### 前言：

这一章主要讲解并发事务的底层实现原理，也就是事务的锁机制，一般比较大的公司，才能在校招面试的时候问到事务实现原理这一层，因为这部分内容算是比较复杂的，但是大家在图论Java学习的工程中，并发编程的知识学了很多，原理都是相通的，所以借助下面的内容描述，好好理解一下MySQL的并发操作。

#### 一 MyISAM表锁：

**MyISAM存储引擎不支持事务处理**，因此它的并发比较简单，只支持到**表锁**的粒度，粒度比较大，并发能力一般，但不会引起死锁的问题，**它支持表共享的读锁**和**表互斥的写锁。**

1. **MyISAM表级锁**

对 MyISAM 表的读操作，不会阻塞其他用户对同一表的读请求，但会阻塞对同一表的写请求；对 MyISAM 表的写操作，则会阻塞其他用户对同一表的读和写操作；MyISAM 表的读操作与写操作之间，以及写操作之间是串行的。

MyISAM 在执行查询语句（SELECT）前，**会自动给涉及的所有表加读锁**，在执行更新操作（UPDATE、DELETE、INSERT 等）前，**会自动给涉及的表加写锁**，这个过程并不需要用户控制，是MySQL Server端自动完成的。

1. **MyISAM并发插入**

mysql> **show variables like 'concurrent\_insert';**

+-------------------+-------+

| Variable\_name | Value |

+-------------------+-------+

| concurrent\_insert | AUTO |

+-------------------+-------+

普通情况下，MyISAM的读操作和写操作都是串行的，但是其实MyISAM也是支持读和写的并发操作的，上面的**concurrent\_insert**变量就是开关，允许一个线程在读的时候，另外一个线程在尾部进行插入（**但是不能并发进行删除delete和更新update**）。

1. **MyISAM的锁调度**

在MyISAM存储引擎下，多个线程并发操作时，线程1视图获取读锁，线程2获取写锁，一般MyISAM认为写操作要比读操作重要，因此线程2几乎都会优先获取写锁，写操作完成后，线程1才会获取读锁。即使线程1的读锁请求先到达，线程2的写锁请求后到达，那么线程2写锁的获取也会排在线程1读锁的前面。

因此，**MyISAM存储引擎不适合大量的更新操作和查询操作，因为查询操作获取读锁的优先级比较低**，会导致客户端查询获取结果的过程很慢。当然MySQL提供了很多参数设置，可以调整读锁的获取优先级，这个大家想深入了解，可以自行网上查询。

#### 二 InnoDB行锁：

**InnoDB 存储引擎支持事务处理，表支持行级锁定，并发能力更好**，InnoDB 实现了以下两种类型的行锁。

**共享锁（S）：**允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。

**排他锁（X)：**允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和排他写锁。

**从上面的描述，可以概括表锁和行锁的特点，如下：**

**表级锁**：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。

**行级锁**：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

InnoDB在实现事务隔离级别的时候，采用的是一种叫做**数据多版本并发控制（MultiVersion Concurrency Control，简称 MVCC 或MCC）机制（当然串行化除外），也经常称为多版本数据库**。MVCC机制会生成一个数据请求时间点的一致性数据快照 （Snapshot)， 并用这个快照来提供一定级别 （语句级或事务级） 的一致性读取。从用户的角度来看，好象是数据库可以提供同一数据的多个版本。

1. InnoDB行锁是通过给索引上的索引项加锁来实现的，而不是给表的行记录加锁实现的，这就意味着，**只有通过索引条件检索数据，InnoDB才使用行级锁，否则InnoDB将使用表锁（因为没有索引嘛，存储引擎只能给所有的行都加锁，和表锁一样，把记录返回给MySQL Server，它会筛选出符合条件的行进行加锁，其余的行就会释放锁）**！

注意：可以用select ... for update语句来进行测试，**for update可以主动获取锁（X排它锁），直到事务提交完成**。

**如下示例，test\_innodb\_lock的id字段没有创建索引**：

mysql> **create table test\_innodb\_lock(id int);**

Query OK, 0 rows affected (0.97 sec)

mysql> **insert into test\_innodb\_lock values(1),(2);**

Query OK, 2 rows affected (0.17 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql> **select \* from test\_innodb\_lock;**

+------+

| id |

+------+

| 1 |

| 2 |

+------+

2 rows in set (0.00 sec)

**按如下示例操作：**

|  |  |
| --- | --- |
| **（MySQL客户端命令行窗口1）事务1** | **（MySQL客户端命令行窗口2）事务2** |
| mysql> set autocommit=0;  Query OK, 0 rows affected (0.00 sec)  mysql> select @@autocommit;  +--------------+  | @@autocommit |  +--------------+  | 0 |  +--------------+  1 row in set (0.00 sec) | mysql> set autocommit=0;  Query OK, 0 rows affected (0.00 sec)  mysql> select @@autocommit;  +--------------+  | @@autocommit |  +--------------+  | 0 |  +--------------+  1 row in set (0.00 sec) |
| mysql> select \* from test\_innodb\_lock where id = 1 for update;  +------+  | id |  +------+  | 1 |  +------+  1 row in set (0.00 sec) | mysql> select \* from test\_innodb\_lock where id = 2 for update;  。。。**等待获取锁当中，证明左边窗口事务1在select id=1的记录时，获取的是表锁，而不是行锁！** |

1. 由于InnoDB的行锁实现是针对索引字段添加的锁，不是针对行记录加的锁，因此虽然访问的是InnoDB引擎下表的不同行，**但是如果使用相同的索引字段作为过滤条件，依然会发生锁冲突**，只能串行进行，不能并发进行。

3）根据前几章学习的内容可知，即使SQL中使用了索引，但是经过MySQL的优化器后，如果认为全表扫描比使用索引效率更高，**此时会放弃使用索引，因此也不会使用行锁，而是使用表锁**，比如对一些很小的表，MySQL就不会去使用索引。

**请大家自行按照上面的示例，验证2,3点。**

#### 三 间隙锁：

当我们用范围条件而不是相等条件检索数据，并请求共享或排他锁时，InnoDB 会给符合条件的已有数据记录的索引项加锁；对于键值在条件范围内但并不存在的记录，叫做“间隙（GAP)” ，InnoDB 也会对这个“间隙”加锁，**这种锁机制就是所谓的间隙锁（Next-Key 锁**）。举例来说， 假如 user 表中只有 101 条记录， 其 userid 的值分别是 1,2,...,100,101， 下面的 SQL：

**Select \* from user where userid > 100 for update;**

是一个范围条件的检索，InnoDB 不仅会对符合条件的 userid 值为 101 的记录加锁，也会对userid 大于 101（但是这些记录并不存在）的“间隙”加锁，**防止其它事务在表的末尾增加数据。**

**InnoDB 使用间隙锁的目的，为了防止幻读**，**以满足串行化隔离级别的要求**，对于上面的例子，要是不使用间隙锁，如果其他事务插入了 userid 大于 100 的任何记录，那么本事务如果再次执行上述语句，就会发生幻读。

#### 四 InnoDB死锁：

MyISAM 表锁是 deadlock free 的， 这是因为 MyISAM 总是一次获得所需的全部锁，要么全部满足，要么等待，**因此不会出现死锁**。但在 InnoDB 中，除单个 SQL 组成的事务外，锁是逐步获得的，即锁的粒度比较小，这就决定了在 InnoDB 中发生死锁是可能的。

|  |  |
| --- | --- |
| **（MySQL客户端命令行窗口1）事务1** | **（MySQL客户端命令行窗口2）事务2** |
| mysql> set autocommit=0;  Query OK, 0 rows affected (0.00 sec)  mysql> create table table1(id int);  Query OK, 0 rows affected (0.59 sec)  mysql> insert into table1 values(1);  Query OK, 1 row affected (0.01 sec) | mysql> set autocommit=0;  Query OK, 0 rows affected (0.00 sec)  mysql> create table table2(id int);  Query OK, 0 rows affected (0.58 sec)  mysql> insert into table2 values(2);  Query OK, 1 row affected (0.02 sec) |
| mysql> select \* from table1 where id = 1 for update;  +------+  | id |  +------+  | 1 |  +------+  1 row in set (0.00 sec) | mysql> select \* from table2 where id = 2 for update;  +------+  | id |  +------+  | 2 |  +------+  1 row in set (0.00 sec) |
| mysql> select \* from table2 where id = 2 for update;  **由于table2的锁已经被事务2获取，所以事务1等待，发生死锁！**  。。。  mysql> select \* from table2 where id = 2 for update;  +------+  | id |  +------+  | 2 |  +------+  1 row in set (4.96 sec) | mysql> select \* from table1 where id = 1 for update;  **由于table1的锁已经被事务1获取，所以事务2等待，发生死锁！**  **mysql> select \* from table1 where id = 1 for update;**  **ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction**  **MySQL自动检测死锁问题，把当前事务回滚，释放该事务持有的锁，此时左边的事务1就能够获取table2的表锁，执行select语句。** |

在上面的例子中，两个事务都需要获得对方持有的排他锁才能继续完成事务，**这种循环锁等待就是典型的死锁**。发生死锁后，InnoDB 一般都能自动检测到，并使一个事务释放锁并回退，另一个事务获得锁，继续完成事务。

这样的死锁问题，一般都是我们自己的应用造成的，和**Java SE多线程死锁的情况相似，大部分都是由于我们多个线程在获取多个锁资源的时候，获取的顺序不同而导致的死锁问题**，因此我们应用在**对数据库的多个表做更新的时候，不同的代码段，应对这些表按相同的顺序进行更新操作，以防止锁冲突导致死锁问题**！

【参考链接】

【美团点评技术团队-Innodb中的事务隔离级别和锁的关系】https://blog.csdn.net/endlu/article/details/51531391

【MySQL事务处理过程 锁和隔离级别 死锁】http://blog.csdn.net/moxigandashu/article/details/62046030