# 7 **行锁功过：怎么减少行锁对性能的影响？**

MySQL的行锁是在引擎层由各个引擎自己实现的，但并不是所以引擎都支持行锁，比如MyISAM引擎就不支持行锁。不支持行锁意味着并发控制只能使用表锁，对于这种引擎的表，同一张表上任何时刻只能有一个更新在进行，这个会影响到业务发展，InnoDB是支持行锁的，这也是MyISAM被InnoDB替代的原因之一。

比如事务A更新了一行，而这时候事务B也要更新同一行，则必须等待事务A的操作完成后才进行更新。

**两阶段锁**

在下面的例子中，事务B的update语句执行时会是什么现象？假设字段id是表t的主键



这个问题的结论取决于事务A在执行完两条update语句后，持有那些锁，以及在什么时候释放。可以验证的是：事务B的update语句会被阻塞，直到事务A执行commit之后，事务B才可以继续执行。

也就是，**在InnoDB事务中，行锁是需要的时候加上，但并不是不需要了就立刻释放了，而是要等到事务结束时才释放，这个就是两阶段锁协议。**

**那就是，如果事务中需要锁多个行，要把最可能造成锁冲突，最可能影响并发度的锁放到尽量往后放，**

假设实现一个电影院在线交易业务，顾客A要在影院B购买电影票。需要以下操作

1. 从顾客A账户余额中扣除电影票价。
2. 给影院B的账户余额增加这张电影票价
3. 记录一条交易日志

要完成这个交易，需要update两条记录，并insert一条记录，当然，为了保证交易的原子性，我们要把三个操作放到一个事务中，那么，要怎样安排这个三个语句在事务中的顺序呢？

试想如果顾客C要在影院B买票，那么这连个事务冲突的部分就是语句2了，因为他需要更新同一影院账户的余额，需要修改同一行数据。

根据两阶段锁协议，不论怎样安排语句顺序，所有的操作需要的行锁都是在事务提交的时候才释放。所以，如果把语句2安排在最后，比如按照3-1-2这样的顺序，那么影院账户余额这一行的锁时间最少。这就是最大程度的减少了事务之间的锁等待，提升了速度。

由于正确的设计，影院余额这一行的锁在一个事务中不会停留很长时间，但是并没有完全解决困扰。

如果影院做活动，可以低价预收一年内的所有的电影票，而且这个活动只做一天，于是在活动时间开始的时候，MySQL就挂了，CPU消耗接近100%，但整个数据库每秒执行不到100个事务，什么原因？

**死锁和死锁检查**

当并发系统中不同的线程出现循环资源依赖，涉及的线程都在等待别的线程是放资源时，就会导致整个几个线程都就进入无限等待的状态，称为死循环，这里用数据库行锁举个例子。



此时，事务A在等待事务B是放id=2的行锁，而事务B在等待事务A是放id=1的行锁，事务A和事务B在互相等待对方是放资源，就是进入了死锁状态，当出现死锁，有两种策略

1. 直接进入等待，直到超时，这个超时时间可以通过参数

Innodb\_lock\_wait\_timeout 来设置

1. 发起死锁检测，主动回滚死锁链条中的一个事务，让其他事务得以继续执行，将参数

Innodb\_deadlock\_detect设置为On，表示开启这个逻辑。

在Innodb中，innodb\_lock\_wait\_timeout的默认值是50s,意味着如果采用第一种策略，当出现死锁，第一个被锁住的线程要过50s才会超时退出，然后其他线程才可能继续执行，对于在线服务来说，这个等待时间是无法接受的。

但是，我们又不可能直接把这个时间设置成一个很小的值，比如1s.这样出现死锁的时候，确实很快就解开了，但是如果不是死锁，而是简单的锁等待？所以，超时时间设置太短的话，会出现很多误伤。

所以，正常情况下使用第二种策略，即：主动死锁检查，而且innodb\_deadlock\_detect的默认值本身就是On.主动死锁检测在发生死锁的时候，是能够快速发现并进行处理的，但是它是有额外负担的。

试想这个过程，每当一个事务被锁的时候，就要看看他所依赖的线程有没有被别人锁住，如此循环，最后判断是否出现循环等待，也就是死锁。

那如果我们上面说到的所有事务都要更新同一行的场景呢？

每个新来的被堵住的线程，都要判断会不会由于自己的加入导致了死锁，这是一个时间复杂度O(n)的操作，假设有1000个并发线程同时更新一行，那么死锁检测就是100万这个量级的。虽然最终检测的结果是没有死锁，但是这期间要消耗大量的CPU资源。因此，CPU利用率很高，但是每秒执行不了几个事务。

根据上面的分析，我们来通论一下，怎么解决这种热点行的更新导致的性能问题？问题的症结在于，死锁检测要消耗大量的CPU资源。

**一种头痛医头的方法，就是如果能确定这个业务一定不会出现死锁，可以临时把死锁检测关闭，**但是这种操作本身带有一定的风险，因为业务设计的时候一般不把死锁当做一个严重的错误，毕竟出现了死锁，就回滚，然后通过业务重试一般就没有问题，这是业务无损的，而关掉死锁检测意味可能出现大量的超时，这是业务有损的。

**另一种是控制并发度，根据上面的分析，**会发现如果并发能够控制，比如同一行同时最多有10个线程控制，那么死锁检测的成本很低，就不会出现这个问题，一个直接的想法是，在客户端控制并发，但是，很快发现这个方法不太可行，因为客户端很多，600个客户端，即使客户端控制到只有5个并发线程，汇总到服务器端以后，峰值并发数也要达到3000.

因此，这个并发控制要做在数据库服务端，如果有消息中间件，可以考虑在中间件实现，如果团队有能修改MySQL源码的人，可以做在MySQL里面，基本思路就是，对于相同的行的更新，在进入引擎之前排队，这样在InnoDB内部就不会有大量的死锁检测工作了。

**如果团队暂时没有数据库方面的饿专家，不能实现这样的方案，怎样优化这个问题？**

可以考虑通过将一行改成逻辑上的多行来减少锁冲突，还是以影院账户为例，可以考虑在多条记录上，比如10个记录，影院账户的总额等于10个记录的和，这样每次要给影院账户加金额的时候，随机选中一个记录来加，这样每次冲突概率编程原来的1/10,可以减少锁等待个数，也就减少了思索检测的CPU消耗。

此方案看上去是无损的，但实际上要根据业务做详细设计，如果账户余额可能会减少，比如退票逻辑，

**讨论题**

如果要删除一个表里面的10000行数据，有一下3种方法可以做到

1. 直接执行delete from T limit 10000;
2. 在一个连接中循环执行20次delete from t limit 500;
3. 在20个连接中同时执行delete from t limit 500;

会选择那一中，为什么？