55 Read Committed隔离级别是如何基于ReadView机制实现的?

图文

 2020-04-15 09:00:24

Read Committed隔离级别是如何基于ReadView机制实现的?

- **如何提问**：每篇文章都有评论区，大家在评论区留言提问
- **如何加入狸猫技术交流群**：
  - 添加微信号：Lvgu0715_ （微信名：绿小九），狸猫技术窝的管理员
  - 发送专栏购买截图
  - 2小时内管理员会拉群，人工操作请耐心等待

今天我们来给大家讲一下，基于之前我们说的ReadView机制是如何实现Read Committed隔离级别的，那么当然了，首先就是要先做一些简单的回顾。所谓的Read Committed隔离级别，我们可以用骚气一点的名字，就是简称为 RC 隔离级别。

这个RC隔离级别，实际上意思就是说你事务运行期间，只要别的事务修改数据还提交了，你就是可以读到人家修改的数据的，所以是会发生不可重复读的问题，包括幻读的问题，都会有的。

那么所谓的ReadView机制，之前我们讲过，他是基于undo log版本链条实现的一套读视图机制，他意思就是说你事务生成一个ReadView，然后呢，如果是你事务自己更新的数据，自己是可以读到的，或者是在你生成ReadView之前提交的事务修改的值，也是可以读取到的。

但是如果是你生成ReadView的时候，就已经活跃的事务，在你生成ReadView之后修改了数据，接着提交了，此时你是读不到的，或者是你生成ReadView以后再开启的事务修改了数据，还提交了，此时也是读不到的。

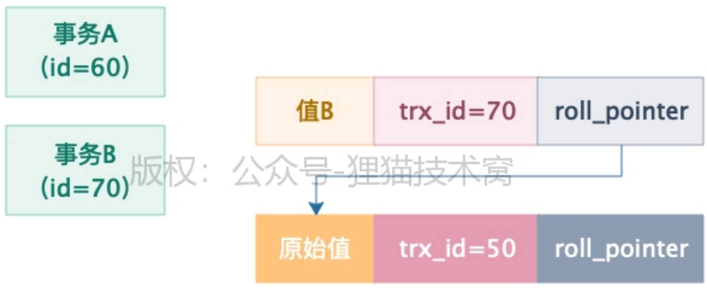
所以上面的那套机制，实际上就是ReadView机制的一个原理。好，那么既然都回顾完了，我们就来看看，如何基于ReadView机制来实现RC隔离级别呢?

其实这里的一个非常核心的要点在于，当你一个事务设置他处于RC隔离级别的时候，他是每次发起查询，都重新生成一个ReadView!

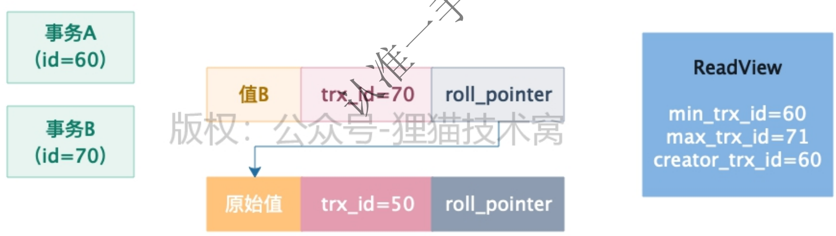
大家注意，这点是非常重要的，接着我们通过画图一步一步来给大家演示这个RC隔离级别是怎么做到的。

首先假设我们的数据库里有一行数据，是事务id=50的一个事务之前就插入进去的，然后现在呢，活跃着两个事务，一个是事务A（id=60），一个是事务B（id=70），此时如下图所示。

现在的情况就是，事务B发起了一次update操作，更新了这条数据，把这条数据的值修改为了值B，所以此时数据的trx_id会变为事务B的id=70，同时会生成一条undo log，由roll_pointer来指向，看下图：

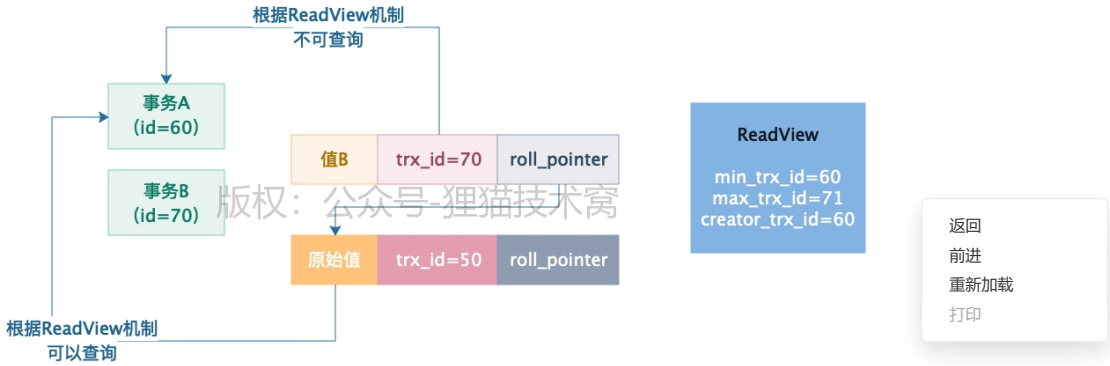


这个时候，事务A要发起一次查询操作，此时他一发起查询操作，就会生成一个ReadView，此时ReadView里的min_trx_id=60，max_trx_id=71，creator_trx_id=60，此时如下图所示。



这个时候事务A发起查询，发现当前这条数据的trx_id是70。也就是说，属于ReadView的事务id范围之间，说明是他生成ReadView之前就有这个活跃的事务，是这个事务修改了这条数据的值，但是此时这个事务B还没提交，所以ReadView的m_ids活跃事务列表里，是有[60, 70]两个id的，所以此时根据ReadView的机制，此时事务A是无法查到事务B修改的值B的。

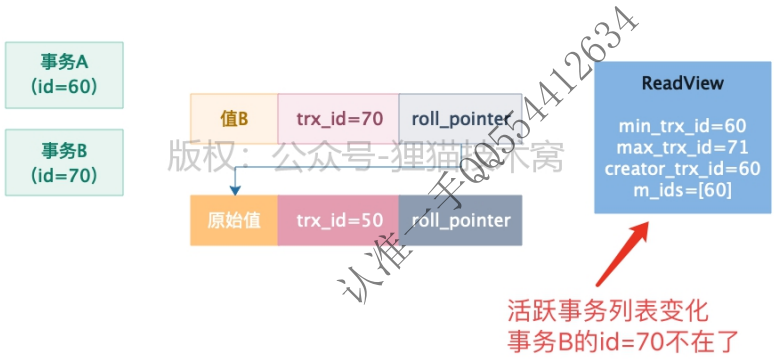
接着就顺着undo log版本链条往下查找，就会找到一个原始值，发现他的trx_id是50，小于当前ReadView里的min_trx_id，说明是他生成ReadView之前，就有一个事务插入了这个值并且早就提交了，因此可以查到这个原始值，如下图。



接着，咱们假设事务B此时就提交了，好了，那么提交了就说明事务B不会活跃于数据库里了，是不是？可以的，大家一定记住，事务B现在提交了。那么按照RC隔离级别的定义，事务B此时一旦提交了，说明事务A下次再查询，就可以读到事务B修改过的值了，因为事务B提交了。

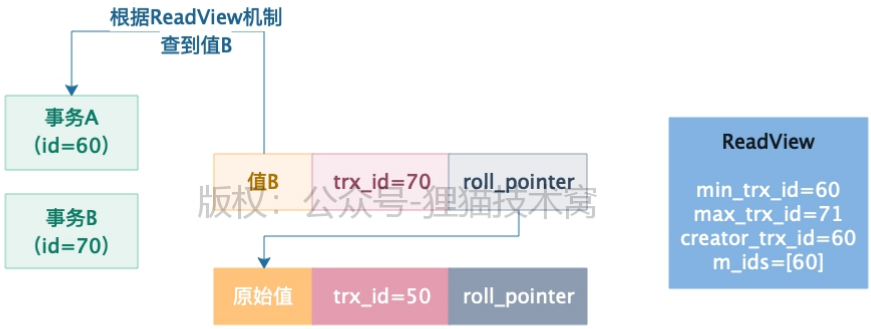
那么到底怎么让事务A能够独到提交的事务B修改过的值呢?

很简单，就是让事务A下次发起查询，再次生成一个ReadView。此时再次生成ReadView，数据库内活跃的事务只有事务A了，因此min_trx_id是60，mac_trx_id是71，但是m_ids这个活跃事务列表里，只会有一个60了，事务B的id=70不会出现在m_ids活跃事务列表里了，如下图。



此时事务A再次基于这个ReadView去查询，会发现这条数据的trx_id=70，虽然在ReadView的min_trx_id和max_trx_id范围之间，但是此时并不在m_ids列表内，说明事务B在生成本次ReadView之前就已经提交了。

那么既然在生成本次ReadView之前，事务B就已经提交了，就说明这次你查询就可以查到事务B修改过的这个值了，此时事务A就会查到值B，如下图所示。



到此为止，RC隔离级别如何实现的，大家应该就理解了，他的关键点在于每次查询都生成新的ReadView，那么如果在你这次查询之前，有事务修改了数据还提交了，你这次查询生成的ReadView里，那个m_ids列表当然不包含这个已经提交的事务了，既然不包含已经提交的事务了，那么当然可以读到人家修改过的值了。

这就是基于ReadView实现RC隔离级别的原理，希望大家好好仔细去体会，实际上，基于undo log多版本链条以及ReadView机制实现的多事务并发执行的RC隔离级别、RR隔离级别，就是数据库的MVCC多版本并发控制机制。

他本质是协调你多个事务并发运行的时候，并发的读写同一批数据，此时应该如何协调互相的可见性。