图文 54 基于undo log多版本链条实现的ReadView机制，到底是什么?

详情 评论

基于undo log多版本链条实现的ReadView机制，到底是什么?

- **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

接着上次我们讲过的undo log多版本链条，我们来讲讲这个基于undo log多版本链条实现的ReadView机制

把这个机制讲明白了，下一次我们再正式讲解RC和RR隔离级别下的MVCC多版本并发控制机制，就很容易理解了。

这个ReadView呢，简单来说，就是你执行一个事务的时候，就给你生成一个ReadView，里面比较关键的东西有4个

- 一个是m_ids，这个就是说此时有哪些事务在MySQL里执行还没提交的；
- 一个是min_trx_id，就是m_ids里最小的值；
- 一个是max_trx_id，这是说mysql下一个要生成的事务id，就是最大事务id；
- 一个是creator_trx_id，就是你这个事务的id

那么现在我们来举个例子，让大家通过例子来理解这个ReadView是怎么用的

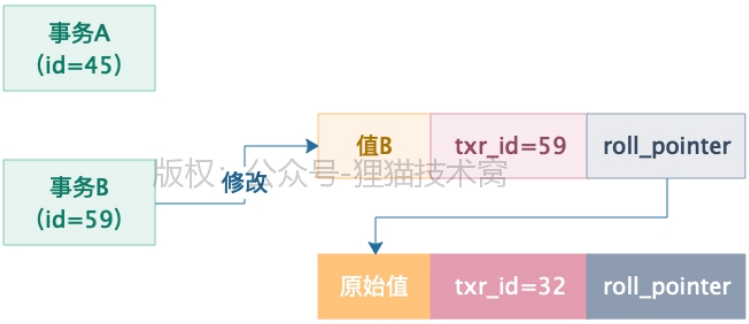
假设原来数据库里就有一行数据，很早以前就有事务插入过了，事务id是32，他的值就是初始值，如下图所示。

接着呢，此时两个事务并发过来执行了，一个是事务A（id=45），一个是事务B（id=59），事务B是要去更新这行数据的，事务A是要去读取这行数据的值的，此时两个事务如下图所示。

现在事务A直接开启一个ReadView，这个ReadView里的m_ids就包含了事务A和事务B的两个id，45和59，min_trx_id就是45，max_trx_id就是60，creator_trx_id就是45，是事务A自己。

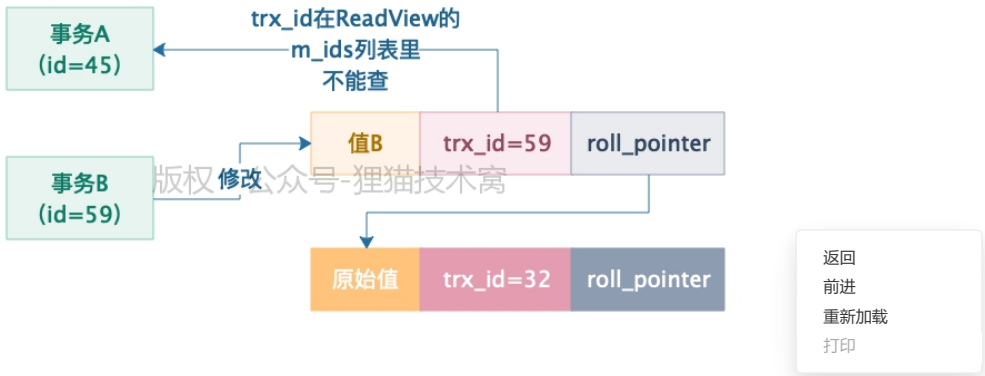
这个时候事务A第一次查询这行数据，会走一个判断，就是判断一下当前这行数据的txr_id是否小于ReadView中的min_trx_id，此时发现txr_id=32，是小于ReadView里的min_trx_id就是45的，说明你事务开启之前，修改这行数据的事务早就提交了，所以此时可以查到这行数据，如下图所示。

接着事务B开始动手了，他把这行数据的值修改为了值B，然后这行数据的txr_id设置为自己的id，也就是59，同时roll_pointer指向了修改之前生成的一个undo log，接着这个事务B就提交了，如下图所示。



这个时候事务A再次查询，此时查询的时候，会发现一个问题，那就是此时数据行里的txr_id=59，那么这个txr_id是大于ReadView里的min_trx_id(45)，同时小于ReadView里的max_trx_id（60）的，说明更新这条数据的事务，很可能就跟自己差不多同时开启的，于是会看一下这个txr_id=59，是否在ReadView的m_ids列表里?

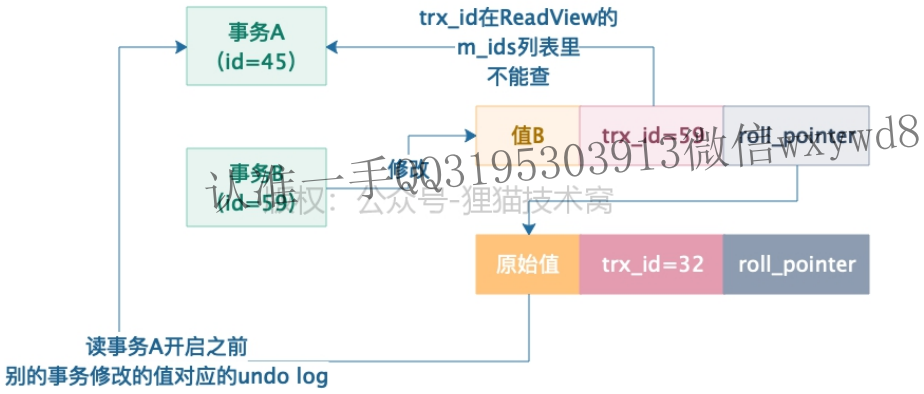
果然，在ReadView的m_ids列表里，有45和59两个事务id，直接证实了，这个修改数据的事务是跟自己同一时段并发执行然后提交的，所以对这行数据是不能查询的！如下图所示。



那么既然这行数据不能查询，那查什么呢？

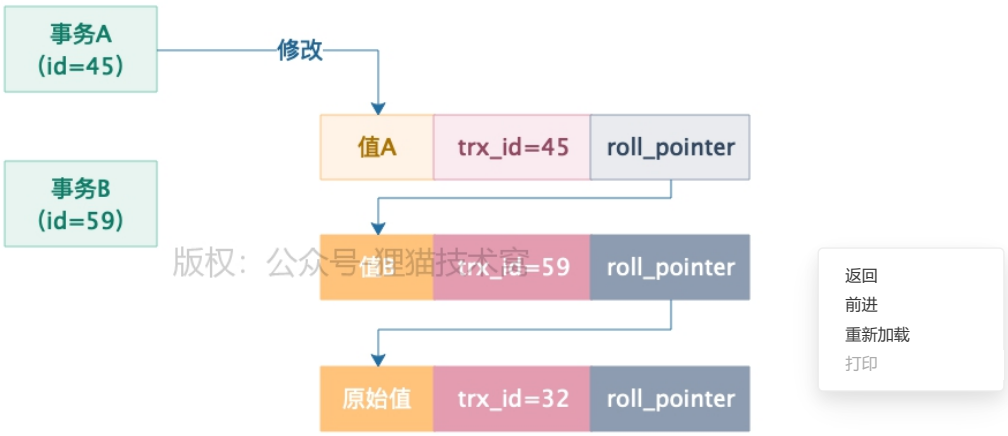
简单，顺着这条数据的roll_pointer顺着undo log日志链条往下找，就会找到最近的一条undo log，trx_id是32，此时发现trx_id=32，是小于ReadView里的min_trx_id（45）的，说明这个undo log版本必然是在事务A开启之前就执行且提交的。

好了，那么就查询最近的那个undo log里的值好了，这就是undo log多版本链条的作用，他可以保存一个快照链条，让你可以读到之前的快照值，如下图。



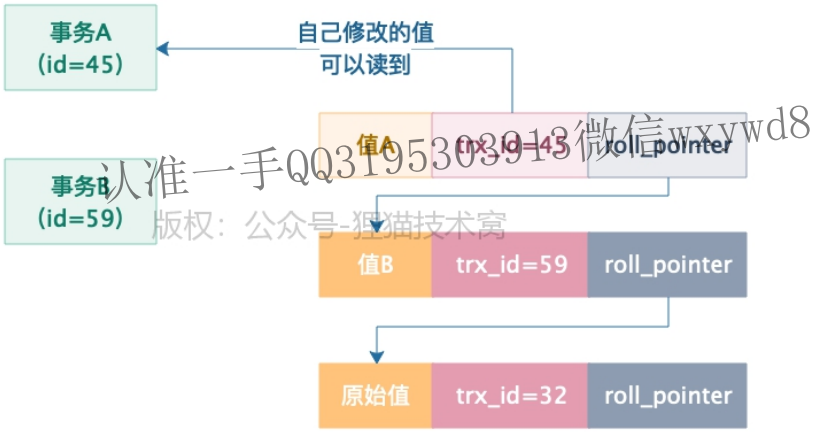
看到这里，大家有没有觉得很奇妙？多个事务并发执行的时候，事务B更新的值，通过这套**ReadView+undo log**日志链条的机制，就可以保证事务A不会读到并发执行的事务B更新的值，只会读到之前最早的值。

接着假设事务A自己更新了这行数据的值，改成值A，trx_id修改为45，同时保存之前事务B修改的值的快照，如下图所示。

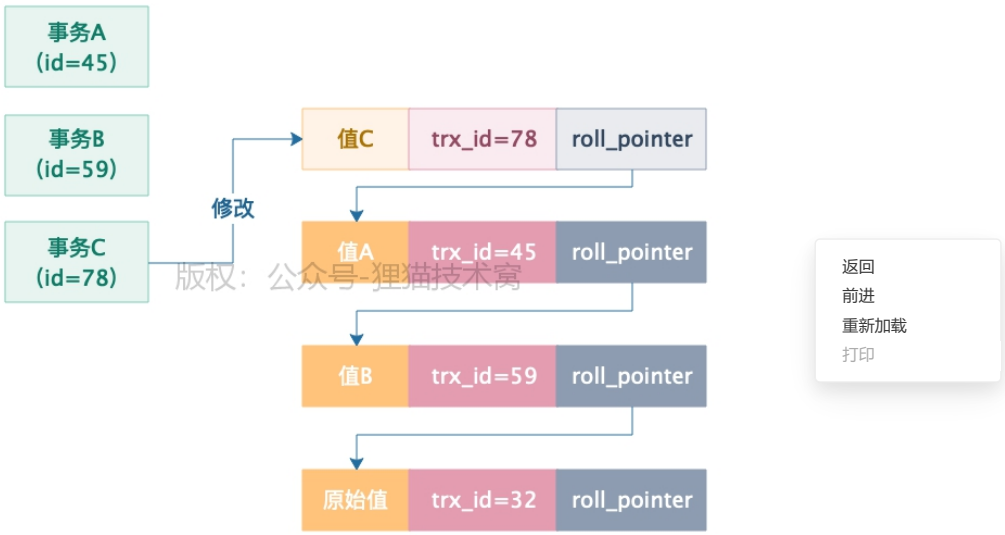


此时事务A来查询这条数据的值，会发现这个trx_id=45，居然跟自己的ReadView里的creator_trx_id（45）是一样的，说明什么？

说明这行数据就是自己修改的啊！自己修改的值当然是可以看到的了！如下图。

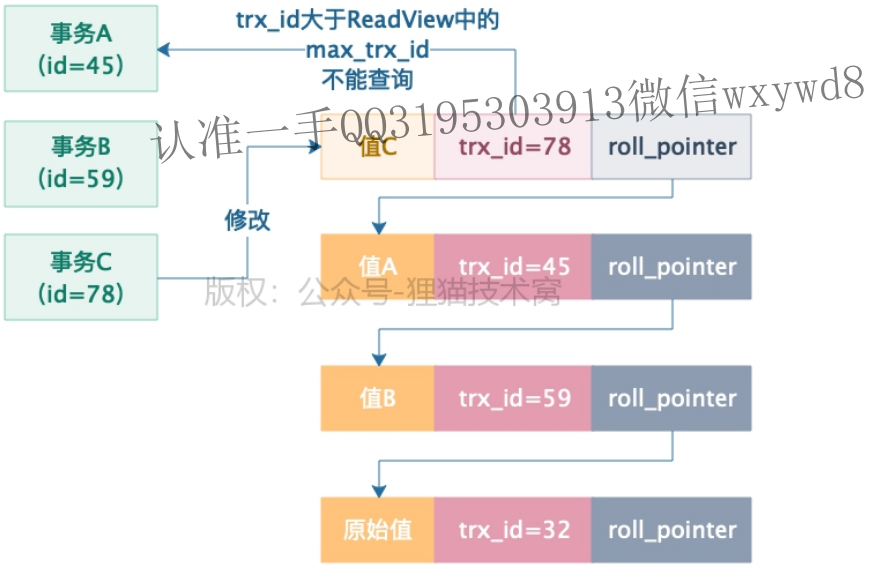


接着在事务A执行的过程中，突然开启了一个事务C，这个事务的id是78，然后他更新了那行数据的值为值C，还提交了，如下图所示。

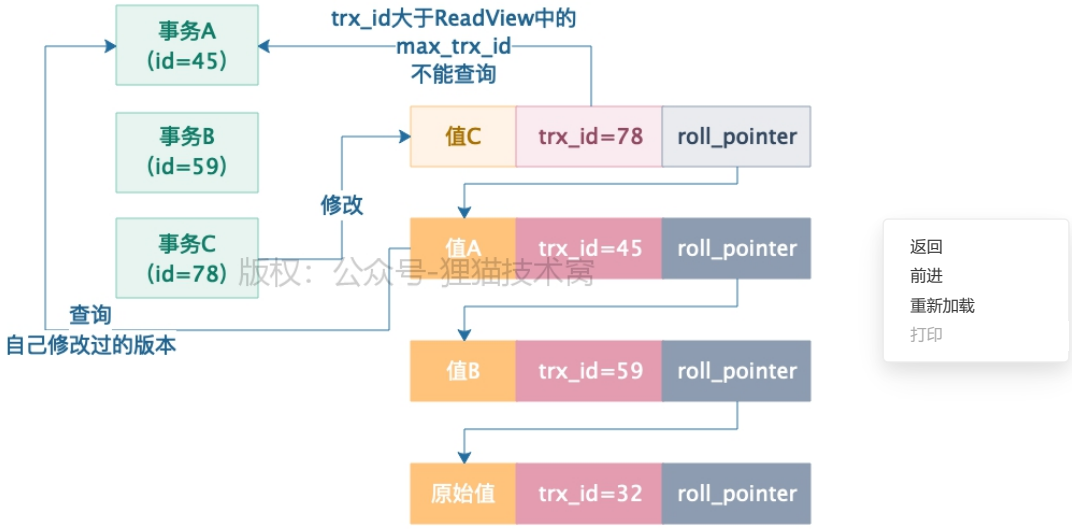


这个时候事务A再去查询，会发现当前数据的trx_id=78，大于了自己的ReadView中的max_trx_id（60），此时说明什么?

说明是这个事务A开启之后，然后有一个事务更新了数据，自己当然是不能看到的了！如下图。



此时就会顺着undo log多版本链条往下找，自然先找到值A自己之前修改的过的那个版本，因为那个trx_id=45跟自己的ReadView里的creator_trx_id是一样的，所以此时直接读取自己之前修改的那个版本，如下图。



不知道大家看到这里感觉如何？通过一系列的图，我相信每个人都能彻底理解这个ReadView的一套运行机制了

通过undo log多版本链条，加上你开启事务时候生产的一个ReadView，然后再有一个查询的时候，根据ReadView进行判断的机制，你就知道你应该读取哪个版本的数据。

而且他可以保证你只能读到你事务开启前，别的提交事务更新的值，还有就是你自己事务更新的值。假如说是你事务开启之前，就有别的事务正在运行，然后你事务开启之后 ，别的事务更新了值，你是绝对读不到的！或者是你事务开启之后，比你晚开启的事务更新了值，你也是读不到的！

通过这套机制就可以实现多个事务并发执行时候的数据隔离，下一次我们继续深入讲解RC和RR两个隔离级别下，这个ReadView是如何运用的。

End