56 MySQL最牛的RR隔离级别，是如何基于ReadView机制实现的?

MySQL最牛的RR隔离级别，是如何基于ReadView机制实现的?

今天来接着给大家讲解，MySQL中最牛的RR隔离级别，是如何同时避免不可重复读问题和幻读问题的。

其实大家现在应该都知道，在MySQL中让多个事务并发运行的时候能够互相隔离，避免同时读写一条数据的时候有影响，是依托undo log版本链条和ReadView机制来实现的。

上次我们都讲过了，基于ReadView机制可以实现RC隔离级别，即你每次查询的时候都生成一个ReadView，这样的话，只要在你这次查询之前有别的事务提交了，那么别的事务更新的数据，你是可以看到的。

那么如果是RR级别呢？RR级别下，你这个事务读一条数据，无论读多少次，都是一个值，别的事务修改数据之后哪怕提交了，你也是看不到人家修改的值的，这就避免了不可重复读的问题。

同时如果别的事务插入了一些新的数据，你也是读不到的，这样你就可以避免幻读的问题。

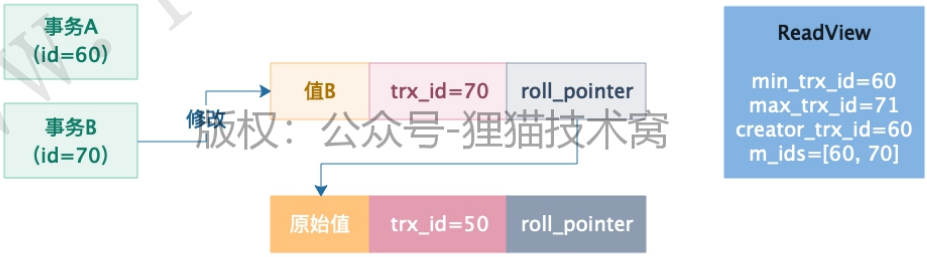
那么到底是如何实现的呢？我们今天来看看。

首先我们还是假设有一条数据是事务id=5的一个事务插入的，同时此时有事务A和事务B同时在运行，事务A的id是60，事务B的id是70，如下图所示。

这个时候，事务A发起了一个查询，他就是第一次查询就会生成一个ReadView，此时ReadView里的creator_trx_id是60，min_trx_id是60，max_trx_id是71，m_ids是[60, 70]，此时ReadView如下图所示。

这个时候事务A基于这个ReadView去查这条数据，会发现这条数据的trx_id为50，是小于ReadView里的min_trx_id的，说明他发起查询之前，早就有事务插入这条数据还提交了，所以此时可以查到这条原始值的，如下图。

接着就是事务B此时更新了这条数据的值为值B，此时会修改trx_id为70，同时生成一个undo log，而且关键是事务B此时他还提交了，也就是说此时事务B已经结束了，如下图所示。



这个时候大家思考一个问题，ReadView中的m_ids此时还会是60和70吗？

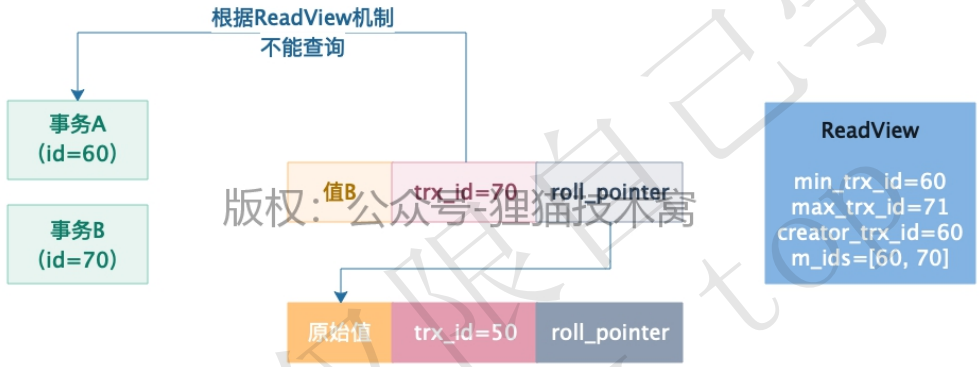
那必然是的，因为ReadView一旦生成了就不会改变了，这个时候虽然事务B已经结束了，但是事务A的ReadView里，还是会有60和70两个事务id。

他的意思其实就是，在你事务A开启查询的时候，事务B当时是在运行的，就是这个意思。

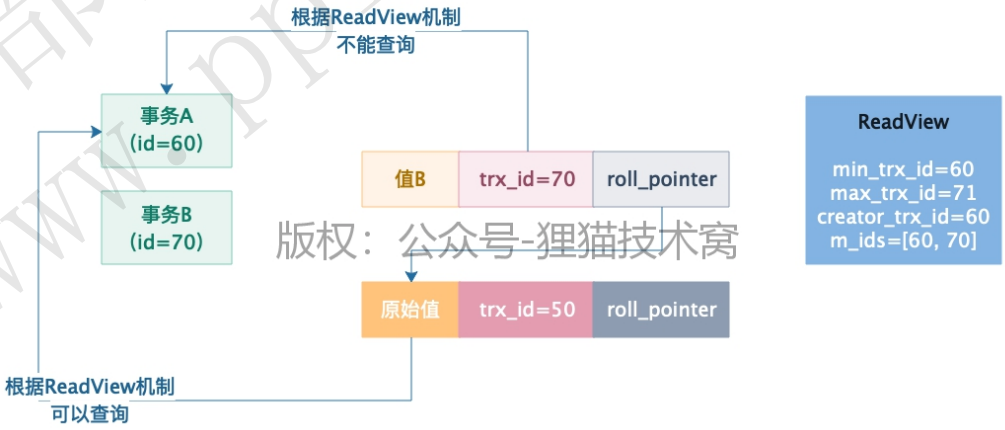
那么好，接着此时事务A去查询这条数据的值，他会惊讶的发现此时数据的trx_id是70了，70一方面是在ReadView的min_trx_id和max_trx_id的范围区间的，同时还在m_ids列表中

这说明什么？

说明起码是事务A开启查询的时候，id为70的这个事务B还是在运行的，然后由这个事务B更新了这条数据，所以此时事务A是不能查询到事务B更新的这个值的，因此这个时候继续顺着指针往历史版本链条上去找，如下图。



接着事务A顺着指针找到下面一条数据，trx_id为50，是小于ReadView的min_trx_id的，说明在他开启查询之前，就已经提交了这个事务了，所以事务A是可以查询到这个值的，此时事务A查到的是原始值，如下图。



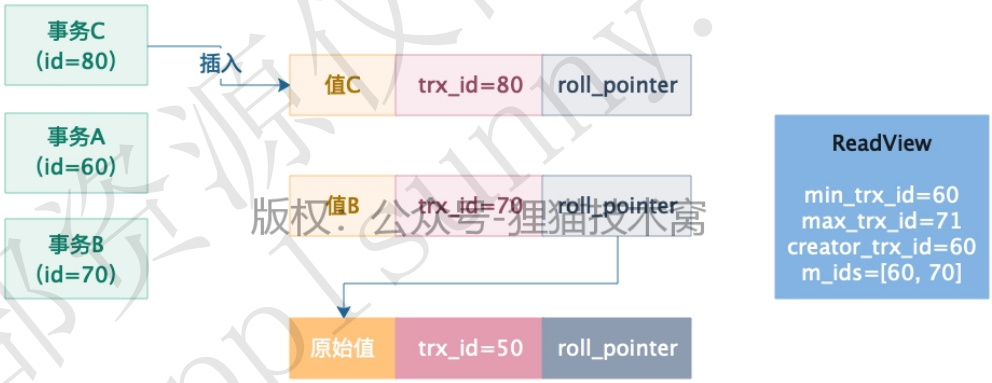
大家看到这里有什么感想？是不是感觉到这一下子就避免了不可重复读的问题？

你事务A多次读同一个数据，每次读到的都是一样的值，除非是他自己修改了值，否则读到的一直会一样的值。

不管别的事务如何修改数据，事务A的ReadView始终是不变的，他基于这个ReadView始终看到的值是一样的！

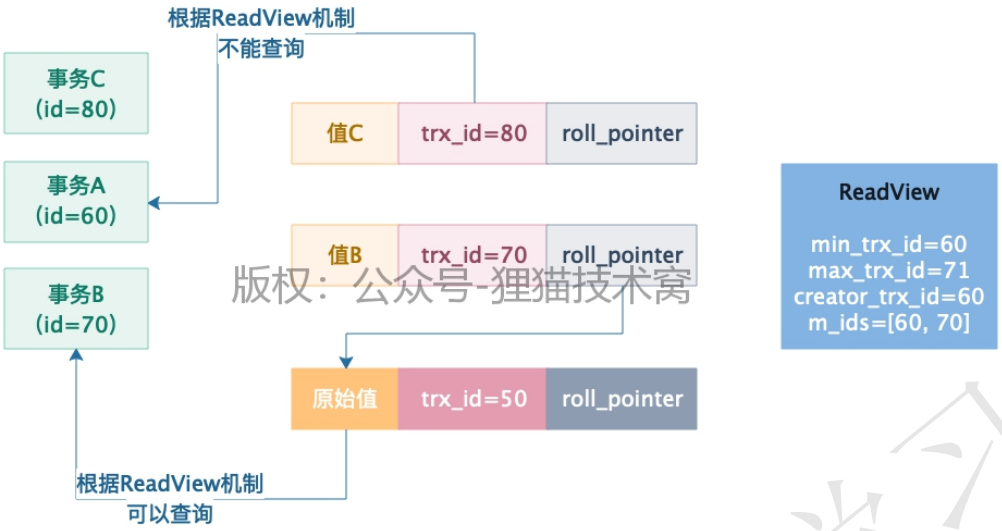
接着我们来看看幻读的问题他是如何解决的。假设现在事务A先用select * from x where id>10来查询，此时可能查到的就是一条数据，而且读到的是这条数据的原始值的那个版本，至于原因，上面都解释过了，如下图。

现在有一个事务C插入了一条数据，然后提交了，此时如下图所示。



接着，此时事务A再次查询，此时会发现符合条件的有2条数据，一条是原始值那个数据，一条是事务C插入的那条数据，但是事务C插入的那条数据的trx_id是80，这个80是大于自己的ReadView的max_trx_id的，说明是自己发起查询之后，这个事务才启动的，所以此时这条数据是不能查询的。

因此事务A本次查询，还是只能查到原始值一条数据，如下图。



所以大家可以看到，在这里，事务A根本不会发生幻读，他根据条件范围查询的时候，每次读到的数据都是一样的，不会读到人家插入进去的数据，这都是依托ReadView机制实现的！

好了，到此为止，如何基于ReadView机制实现RR隔离级别，避免不可重复读问题和幻读问题，就全部讲解清楚了，下次我们来做一个多事务并发隔离机制的总结。

End