在 MySQL 中是如何通过 MVCC 机制来解决不可重复读和幻读问题的?

原创 刘进坤 菜鸟飞呀飞 2020-09-08 00:58

前言

接上篇文章《一文搞懂 undo log 版本链与 ReadView 机制如何让事务读取到该读的数据》，本文接下来介绍在可重复读隔离级别下，MySQL 是如何解决不可重复读和幻读问题的？

本文的内容严重依赖上篇文章的知识，建议读者先阅读上篇文章。

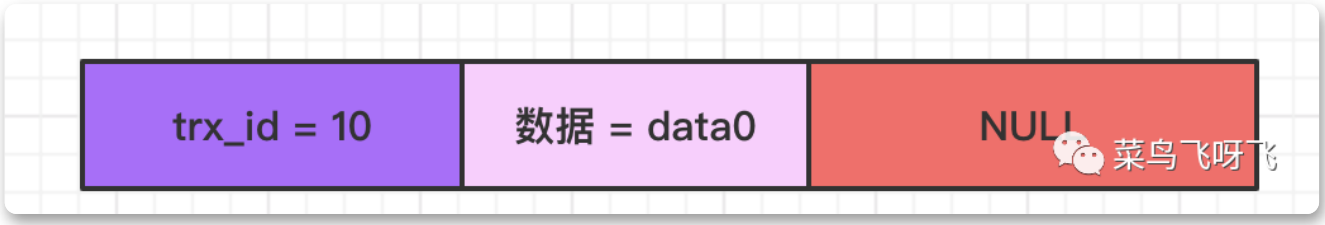
不可重复读

「不可重复读现象指的是，在一个事务内，连续两次查询同一条数据，查到的结果前后不一样」。

在 MySQL 的可重复读隔离级别下，不存在不可重复读的问题，那么 MySQL 是如何解决的呢？

答案就是 MVCC 机制。MVCC 是 Mutil-Version Concurrent Control(多版本并发控制)的缩写，它指的是数据库中的每一条数据，会存在多个版本。对同一条数据而言，MySQL 会通过一定的手段（ReadView 机制）控制每一个事务看到不同版本的数据，这样也就解决了不可重复读的问题。

假设现有一条数据，它的 row_trx_id=10，数据的值为 data0，它的 roll_pointer 指针为 null。

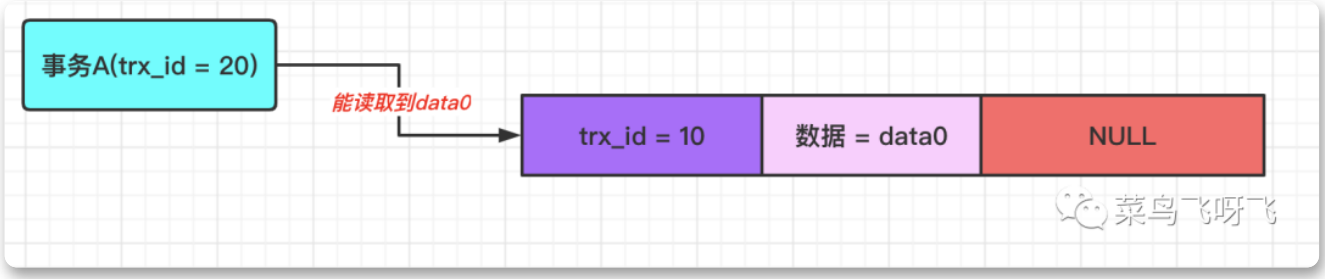


假设现在有事务 A 和事务 B 并发执行，事务 A 的事务 id 为 20，事务 B 的事务 id 为 30。

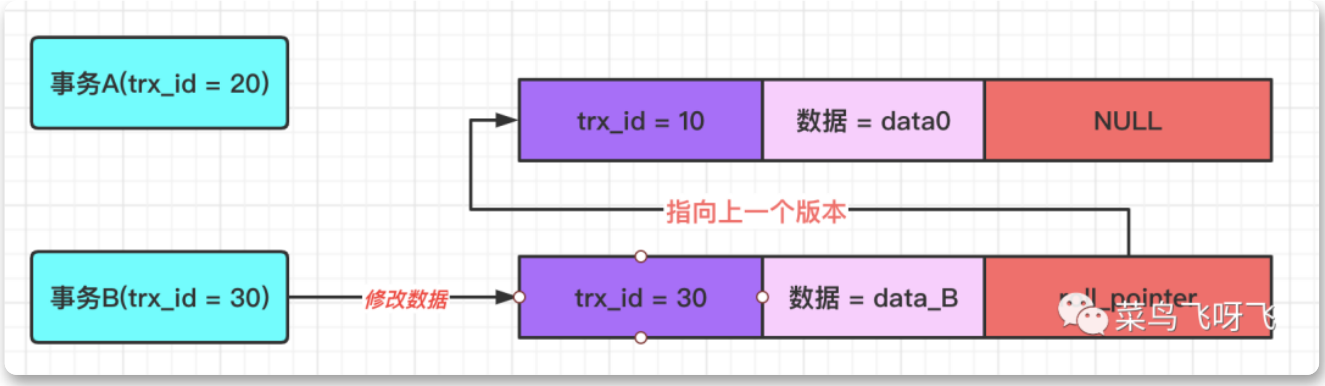
现在事务 A 开始第一次查询数据，那么此时 MySQL 会为事务 A 产生一个 ReadView，此时 ReadView 的内容如下：m_ids=[20,30]，min_trx_id=20，max_trx_id=31，creator_trx_id=20。

此时由于数据的最新版本的 row_trx_id=10，**「小于事务 A 的 ReadView 中的 min_trx_id，这表明这个版本的数据是在事务 A 开启之前就提交的」**，因此事务 A 可以读取到数据，读取到的值为 data0。

「结论：事务 A 第一次查询到的数据为 data0」



接着事务 B(trx_id=30)去修改数据，将数据修改为 data_B，并提交事务，此时 MySQL 会写一条对应的 undo log，数据就会新增一个版本，undo log 版本就变成了如下图所示的结构，数据的最新版本的 row_trx_id 就是事务 B 的事务 id，即：30。



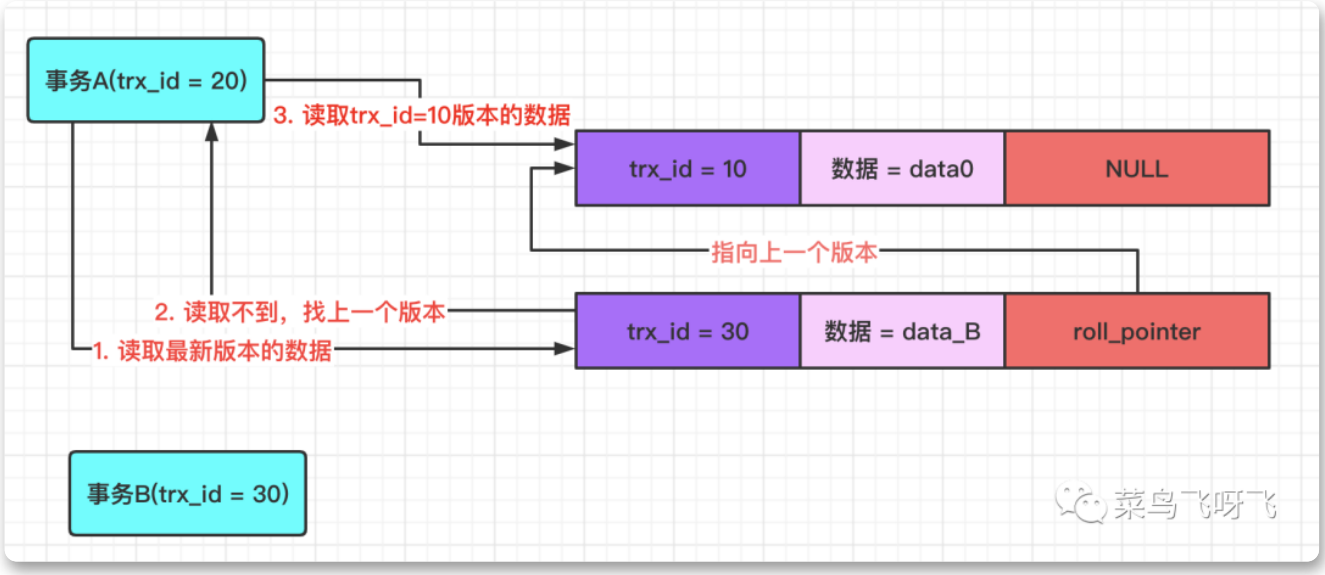
此时，事务 B 已经提交了，因此系统中活跃事务的数组里就没有 30 这个 id 了。

「重点来了，事务 A 的 ReadView 是在发起第一次查询的时候创建的，当时系统中的活跃事务有 20 和 30 这两个 id，那么此时当事务 B 提交以后，事务 A 的 ReadView 的 m_ids 会变化吗？不会。因为是可重复读隔离级别下，对于读事务，只会在事务查询的第一次创建 ReadView，后面的查询不会再重新创建」

接着事务 A(trx_id=20)开始第二次查询数据，前面事务 A 已经创建了 ReadView，所以在第二次查询时，不会再重复创建 ReadView 了。

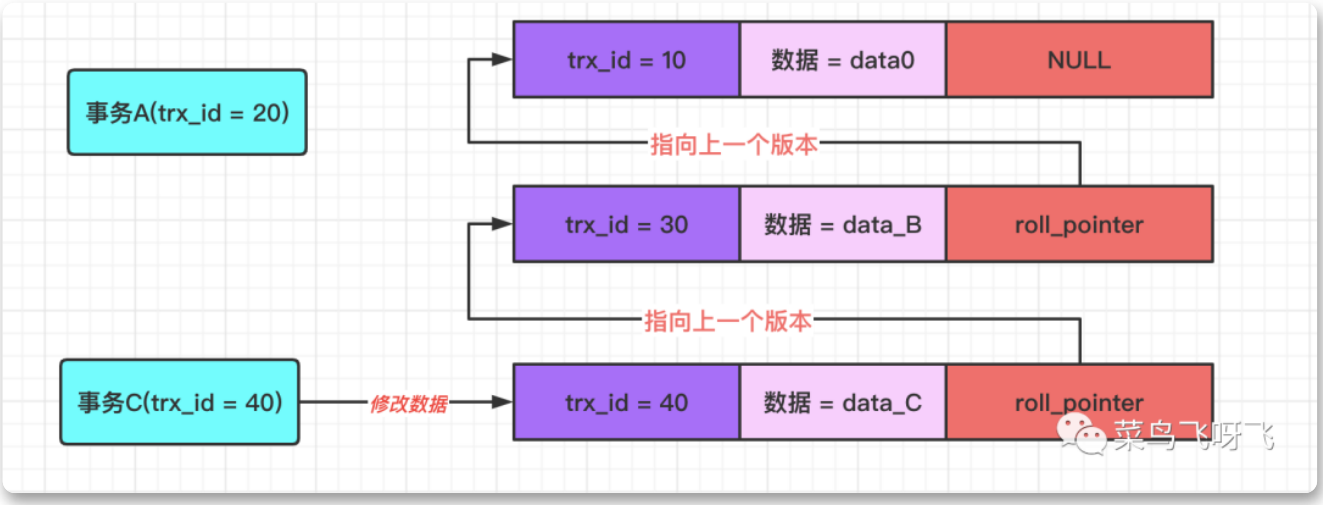
此时在 undo log 版本链中，数据最新版本的事务 id 为 30，根据 ReadView 机制（什么是 ReadView 机制，可以去阅读上一篇文章），发现 30 处于事务 A 的 ReadView 中 min_trx_id 和 max_trx_id 之间，因此还需要判断 30 是否处于 m_ids 数组内，结果发现 30 确实在 m_ids 数组中，**「这就表示这个版本的数据是和自己在同一时刻开启事务所提交的，因此不能让自己读取。」**

所以此时事务 A 需要沿着 undo log 版本链继续向前找，最终发现 row_id=10 的版本数据自己可以读取到，因此事务 A 查询到的值是 data0。



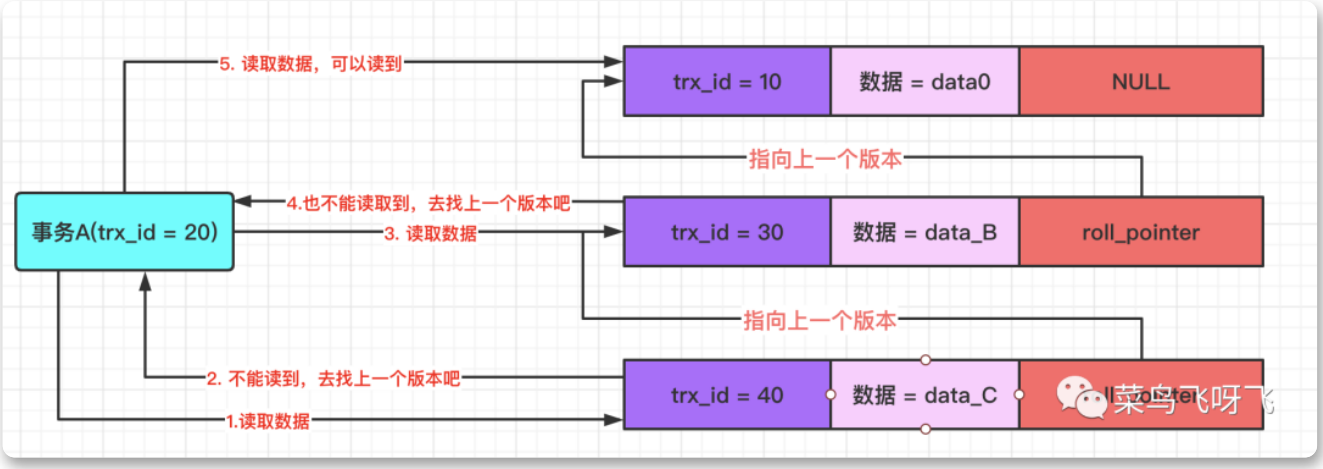
「结论：事务 A 第二次查询到的数据为 data0。这与事务 A 第一次查询的数据结果相同，没有出现不可重复读的现象。」

那假设后来又创建了一个事务 C，id 为 40，并且事务 C 将数据修改为了 data_C。然后数据的 undo log 版本链变为了如下如所示。



然后事务 A 发起第三次查询，此时事务 A 仍然不会再重新创建 ReadView，所以此时它的 ReadView 依旧是：m_ids=[20,30]，min_trx_id=20，max_trx_id=31，creator_trx_id=20。

由于数据最新的版本的为 trx_id=40，依照 ReadView 机制，40 大于事务 A 中的 max_trx_id，**「这表示这是在事务 A 开启之后的事务提交的数据，因此事务 A 不能读取到」**，所以需要沿着 undo log 版本链往前找，然而 trx_id=30 的版本事务 A 也不能读到，继续向前找，最终读取到 trx_id=10 的版本数据，即 data0。



这样，在事务 A 内，一共发起了 3 次查询，每次查询的数据都是 data0，没有出现不可重复读的现象。

幻读

幻读特指后面的查询比前面的查询的记录条数多，看到了前面没看到的数据，就像产生幻觉一样，因此称之为幻读。

快照读与当前读

在解释 MySQL 的可重复读隔离级别解决了幻读问题之前，我们先来看两个定义：**「快照读与当前读」**。

我们知道，在事务开启的时候，会基于当前系统中数据库的数据，为每个事务生成一个快照，也叫做 ReadView，后面这个事务所有的读操作都是基于这个 ReadView 来读取数据，这种读称之为快照读。**「我们在实际的工作中，所使用的 SQL 查询语句基本都是快照读。」**

通过前面介绍的 undo log 版本链，我们知道，每行数据可能会有多个版本，如果每次读取时，**「我们都强制性的读取最新版本的数据，这种读称之为当前读，也就是读取最新的数据」**。什么样的 SQL 查询语句叫做当前读呢？例如在 select 语句后面加上**「for update 或者 lock in share mode」**等。

```
# 加上排他锁
select * from t for update;
# 加上共享锁
select * from t for lock in share mode;
```

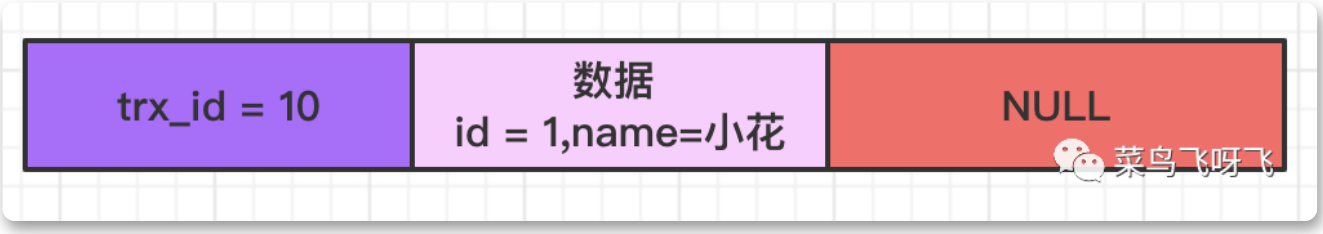
可以发现，当前读的这两种写法，在查询过程中都是需要加锁的，因此它们能读取到最新的数据。

「需要说明的是，在 MySQL 可重复读隔离级别下，幻读问题确实不存在。但是 MVCC 机制解决的是快照读的幻读问题，并不能解决当前读的幻读问题。当前读的幻读问题是通过间隙锁解决的，至于什么是间隙锁，以后的文章中会介绍，有兴趣的读者可以自己去了解。」

因此，**「本文的后半部分，全部是基于快照读来进行解释的」**。

如何解决幻读

假设现在表 t 中只有一条数据，数据内容中，主键 id=1，隐藏的 trx_id=10，它的 redo log 如下图所示。



假设现在有事务 A 和事务 B 并发执行，事务 A 的事务 id 为 20，事务 B 的事务 id 为 30。

现在事务 A 开始第一次查询数据，查询的 SQL 语句如下。

```
select * from where id >= 1;
```

在开始查询之前，MySQL 会为事务 A 产生一个 ReadView，此时 ReadView 的内容如下：m_ids=[20,30]，min_trx_id=20，max_trx_id=31，creator_trx_id=20。

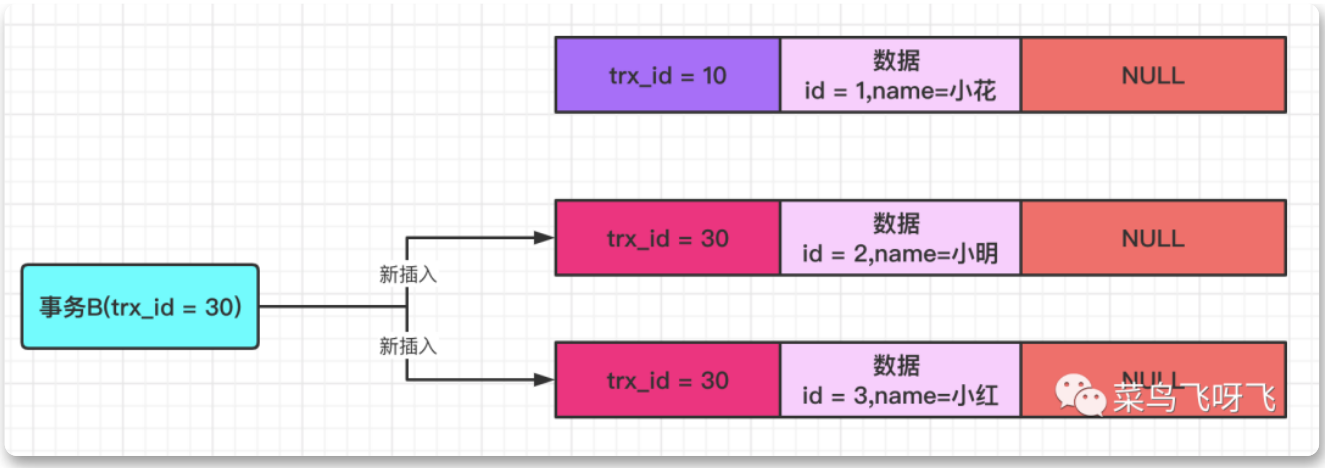
由于此时表 t 中只有一条数据，且符合 where id>=1 条件，因此会查询出来。**「然后通过 ReadView 机制，发现该行数据的 row_id=10，小于事务 A 的 ReadView 里 min_trx_id，这表示这条数据是事务 A 开启之前，其他事务就已经提交了的数据，因此事务 A 可以读取到。」**

「结论：事务 A 的第一次查询，能读取到一条数据，id=1。」

接着事务 B(trx_id=30)，往表 t 中新插入两条数据，SQL 语句如下。

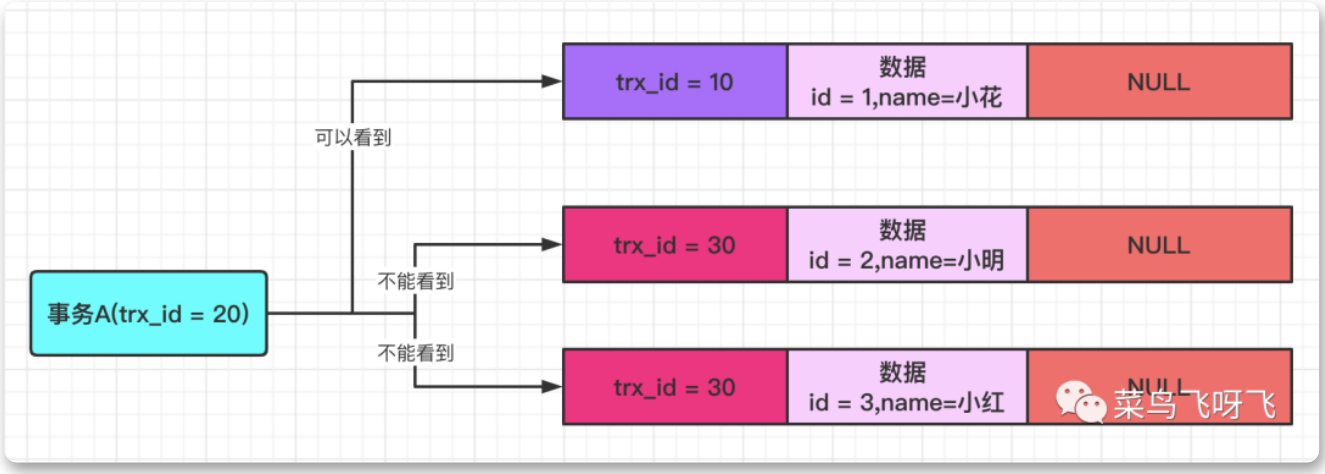
```
insert into t(id,name) values(2,'小明');
insert into t(id,name) values(3,'小红');
```

然后事务提交事务，那么此时表 t 中就有三条数据了，对应的 undo 如下图所示：



接着事务 A 开启第二次查询，根据可重复读隔离级别的规则，此时事务 A 并不会再重新生成 ReadView。此时表 t 中的 3 条数据都满足 where id>=1 的条件，因此会先查出来，然后再根据 ReadView 机制，判断每条数据是不是都可以被事务 A 看到。

1. 首先 id=1 的这条数据，前面已经说过了，可以被事务 A 看到。
2. 然后是 id=2 的数据，它的 trx_id=30，此时事务 A 发现，这个值处于 min_trx_id 和 max_trx_id 之间，因此还需要再判断 30 是否处于 m_ids 数组内。由于事务 A 的 m_ids=[20,30]，因此在数组内，这表示 id=2 的这条数据是与事务 A 在同一时刻启动的其他事务提交的，所以这条数据不能让事务 A 看到。
3. 同理，id=3 的这条数据，trx_id 也为 30，因此也不能被事务 A 看见。



「结论：最终事务 A 的第二次查询，只能查询出 id=1 的这条数据。这和事务 A 的第一次查询的结果是一样的，因此没有出现幻读现象，所以说在 MySQL 的可重复读隔离级别下，不存在幻读问题。」

总结

本文结合 ReadView 机制，介绍了 MySQL 在可重复读隔离级别下，是如何解决不可重复读和幻读问题的，其核心点在于 ReadView 的原理，以及在可重复读隔离级别下，如果事务只是进行查询操作，那么就**「只会在第一次查询的时候生成 ReadView 快照，这一点和读提交隔离级别是最大的区别」**。

同时，文中还简单介绍了快照读和当前读的区别，快照读指的是基于 ReadView 读取数据，当前读指的是读取数据的最新版本。

另外，需要注意的是，文中只是介绍了 MVCC 如何解决快照读的幻读问题，而当前读的幻读问题，则是通过间隙锁来解决的。