我以为我对Mysql事务很熟，直到我遇到了阿里面试官

程序员面试 6月5日

以下文章来源于非科班的科班 ，作者黎杜

迎面走来了一个风尘仆仆的身穿格子衫的男子，手里拿着一个MacBook Pro，看着那稀少的发量，和那从容淡定的眼神。

我心里一颤，我去，这是架构师，架构师来面我技术面，我心里顿时不淡定了，表面很稳实则心里慌得一批。

果然，他手里拿着我的简历，快速的扫了一下，然后用眼角余光看了一下我，上来就开问。

面试官： 看你简历上说精通Mysql优化方法，你先来说说你对Mysql的事务的了解吧。

我心里喜了一下，这个简单啊，哥我可是北大(背大)的，再来面试之前，早就有准备的，二话不说，上去就是背。

我： 好的，数据库的事务是指一组sql语句组成的数据库逻辑处理单元，在这组的sql操作中，要么全部执行成功，要么全部执行失败。

我： 这里的一组sql操作，举个简单又经典的例子就是转账了，事务A中要进行转账，那么转出的账号要扣钱，转入的账号要加钱，这两个操作都必须同时执行成功，为了确保数据的一致性。

面试官： 刚才你提到了数据一致性，你知道事务的特性吗？说说你的理解。

我： 在Mysql中事务的四大特性主要包含：**原子性（Atomicity）**、**一致性（Consistent）**、**隔离性（Isalotion）**、**持久性(Durable)**，简称为 `ACID` 。

我： 原子性是指事务的原子性操作，对数据的修改要么全部执行成功，要么全部失败，实现事务的原子性，是基于日志的 `Redo/Undo` 机制。

我： 一致性是指执行事务前后的状态要一致，可以理解为数据一致性。隔离性侧重指事务之间相互隔离，不受影响，这个与事务设置的隔离级别有密切的关系。

我： 持久性则是指在一个事务提交后，这个事务的状态会被持久化到数据库中，也就是事务提交，对数据的新增、更新将会持久化到数据库中。

我： 在我的理解中，原子性、隔离性、持久性都是为了保障一致性而存在的，一致性也是最终的目的。

心里暗自欢喜，背完了，平时背的多，面试就会说，幸好难不倒我。

面试官： 刚才你说原子性是基于日志的 `Redo/Undo` 机制，你能说一说 `Redo/Undo` 机制吗？

啊哈？我都说了什么，不小心给自己埋了一颗大雷。不慌，哥脑子里还有货，假装若有所思的停了几十秒，接着背。

我： Redo/Undo机制比较简单，它们将所有对数据的更新操作都写到日志中。

我： Redo log用来记录某数据块被修改后的值，可以用来恢复未写入 data file 的已成功事务更新的数据；Undo log是用来记录数据更新前的值，保证数据更新失败能够回滚。

我： 假如数据库在执行的过程中，不小心崩了，可以通过该日志的方式，回滚之前已经执行成功的操作，实现事务的一致性。

面试官： 可以举一个场景，说一下具体的实现流程吗？

我： 可以的，假如某个时刻数据库崩溃，在崩溃之前有事务A和事务B在执行，事务A已经提交，而事务B还未提交。当数据库重启进行 crash-recovery 时，就会通过Redo log将已经提交事务的更改写到数据文件，而还没有提交的就通过Undo log进行roll back。

面试官： 之前你还提到事务的隔离级别，你能说一说吗？

我： 可以的，在Mysql中事务的隔离级别分为四大等级，**读未提交（READ UNCOMMITTED）**、**读提交 （READ COMMITTED）**、**可重复读 （REPEATABLE READ）**、**串行化（SERIALIZABLE）**。

我： 读未提交会读到另一个事务的未提交的数据，产生脏读问题，读提交则解决了脏读的，出现了不可重复读，即在一个事务任意时刻读到的数据可能不一样，可能会受到其它事务对数据修改提交后的影响，一般是对于update的操作。

我： 可重复读解决了之前不可重复读和脏读的问题，但是由带来了幻读的问题，幻读一般是针对inser操作。

我： 例如：第一个事务查询一个User表id=100发现不存在该数据行，这时第二个事务又进来了，新增了一条id=100的数据行并且提交了事务。

我： 这时第一个事务新增一条id=100的数据行会报主键冲突，第一个事务再select一下，发现id=100数据行已经存在，这就是幻读。

面试官： 小伙子你能演示一下吗？我不太会你能教教我吗？我电脑在这里，你演示我看一看。

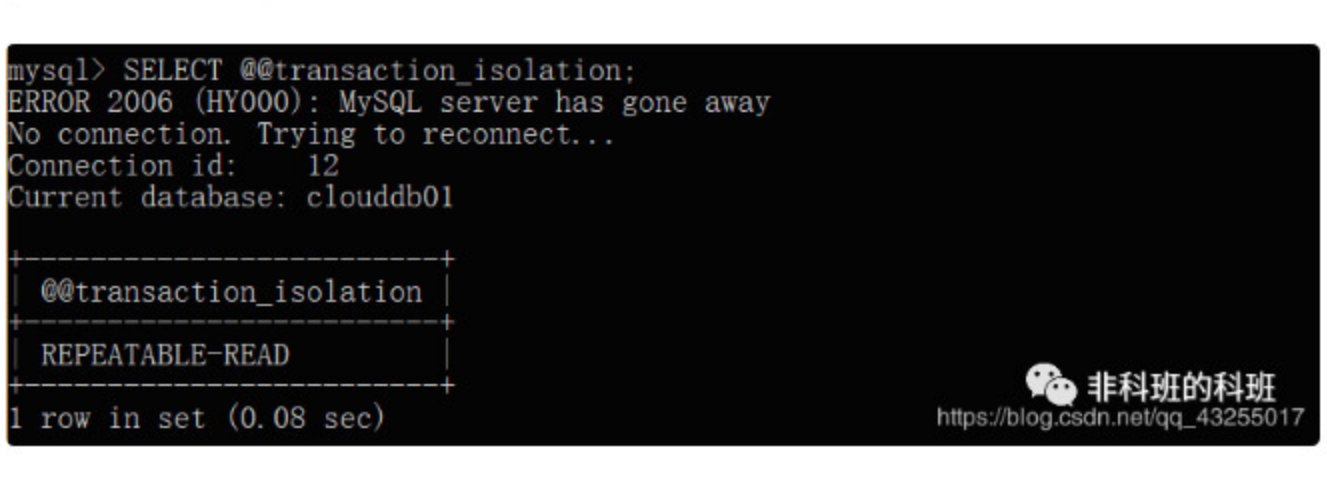
男人的嘴骗人的鬼，我信你个鬼，你这糟老头子坏得很，出来装X总是要还的，只能默默含泪把它敲完。

我： 首先创建一个User表，最为一个测试表，测试表里面有三个字段，并插入两条测试数据。

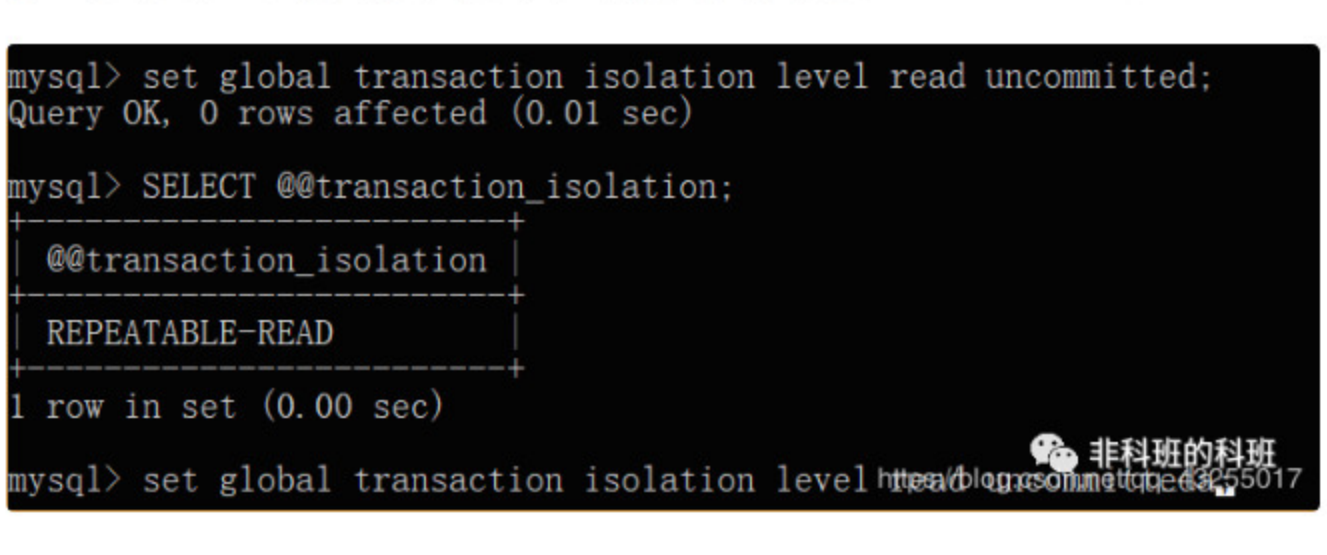
```
CREATE TABLE User (
  id INT(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(20),
  age INT      DEFAULT 0
) ENGINE=InnoDB DEFAULT CHARSET=gb2312;

INSERT INTO `user` VALUES (1, 'zhangsan', 23);
INSERT INTO `user` VALUES (2, 'lisi', 20);
```

我： 在Mysql中可以先查询一下他的默认隔离级别，可以看出Mysql的默认隔离级别是 `REPEATABLE-READ` 。



我： 先来演示一下读未提交，先把默认的隔离级别修改为 `READ UNCOMMITTED` 。



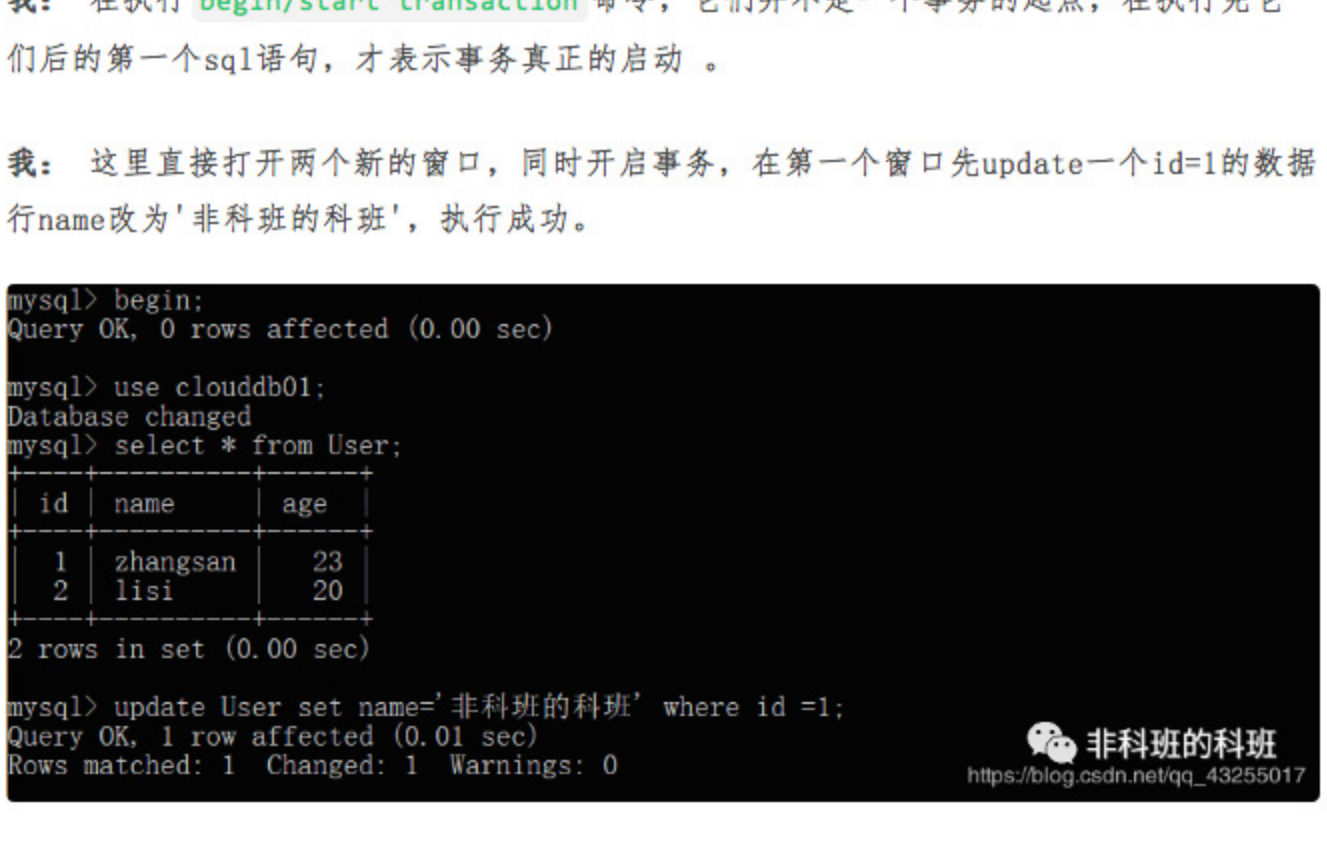
我： 他设置隔离级别的语句中set global transaction isolation level read uncommitted，这里的global也可以换成session，global表示全局的，而session表示当前会话，也就是当前窗口有效。

我： 当设置完隔离级别后对于之前打开的会话，是无效的，要重新打开一个窗口设置隔离级别才生效。

我： 然后是开启事务，Mysql中开启事务有两种方式 `begin/start transaction` ，最后提交事务执行commit，或者回滚事务rollback。

我： 在执行 `begin/start transaction` 命令，它们并不是一个事务的起点，在执行完它们后的第一个sql语句，才表示事务真正的启动 。

我： 这里直接打开两个新的窗口，同时开启事务，在第一个窗口先update一个id=1的数据行name改为'非科班的科班'，执行成功。



我： 然后再第二个窗口执行两次的查询，分别是窗口一update之前的查询和update之后的查询。

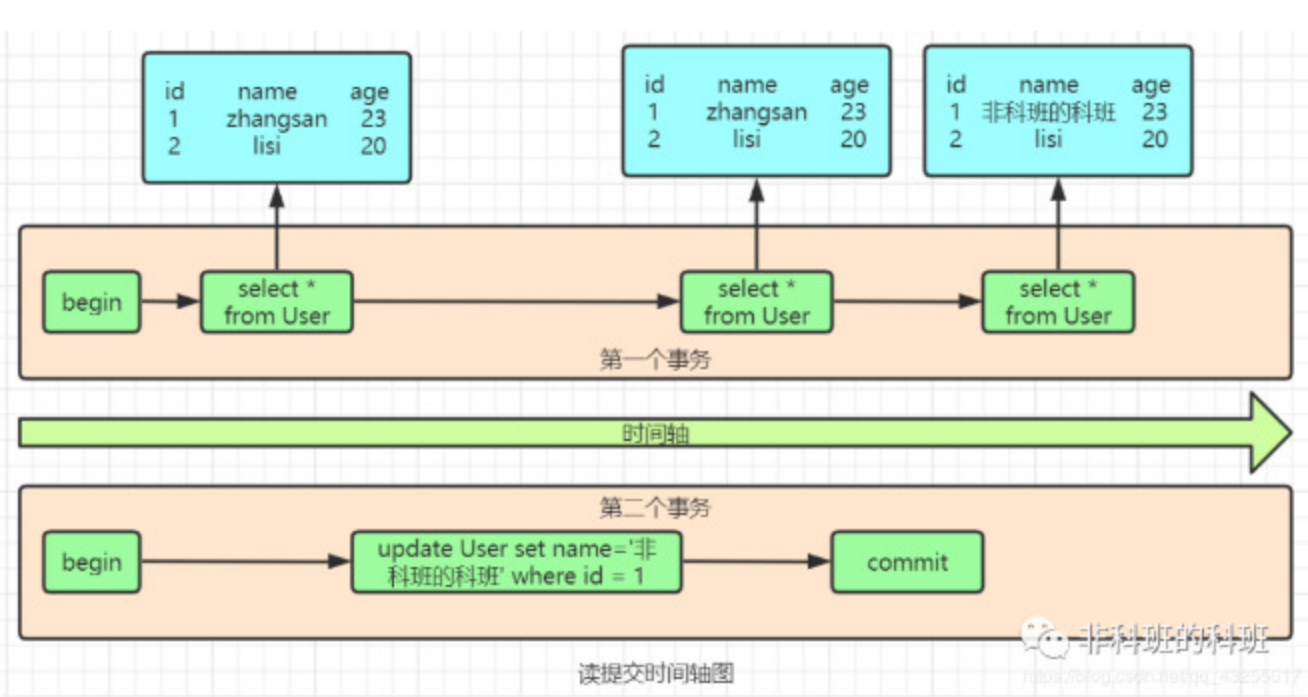
我： 第一个session产生的未提交的事务的状态就会直接影响到第二sesison，也就是脏读。

我： 对于读提交也是一样的，开启事务后，第一个事务先执行查询数据，然后第二个session执行update操作，但是还没有commit，这是第一个session再次select，数据并没有改变，再第二个session执行commit之后，第一个session再次select就是改变后的数据了。

我： 这样第一个事务的查询结果就会收到第二事务的影响，这个也就是产生不可重复读的问题。

面试官： 小伙子你能画一下他执行的过程图吗？你讲的我有点乱，我还没有彻底明白。

我心里一万只什么马在飞过，欲哭无泪，这面试官真难伺候，说时迟那时快，从左屁股兜抽出笔，从右屁股兜拿出纸，开始画。



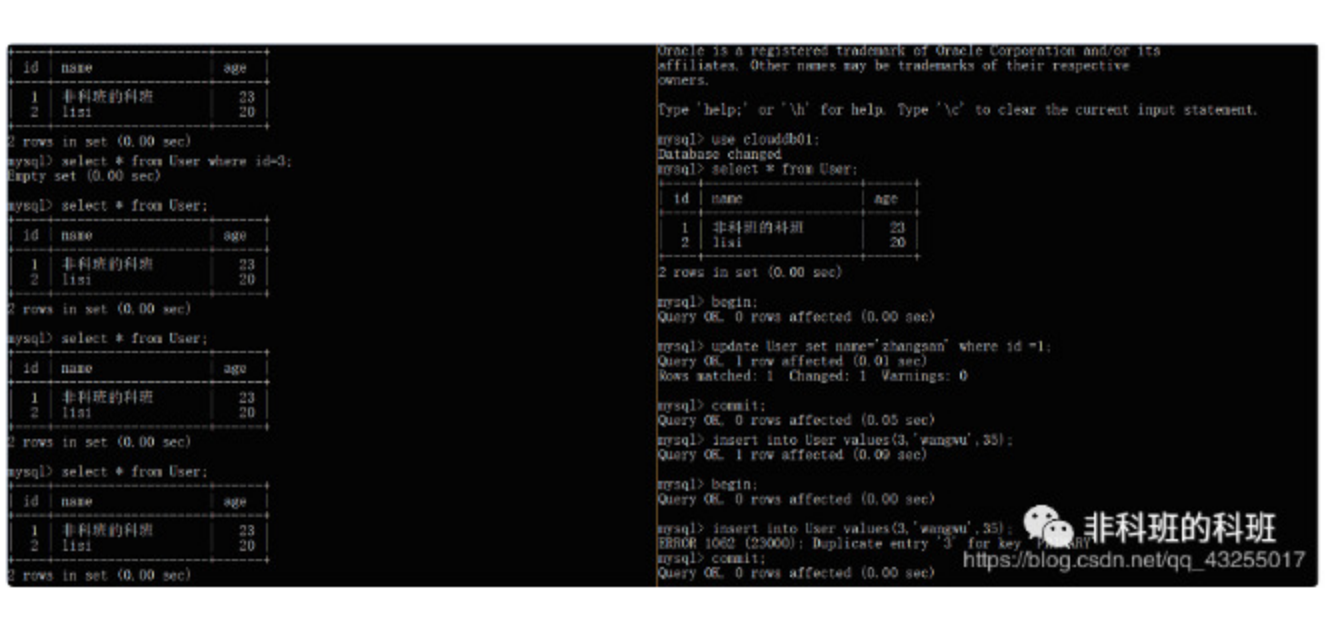
读提交时间轴图

我： 这个是读提交的时间轴图，读未提交的时间轴图，原理也一样的，第二个select的时候数据就已经改变了。

这是面试官拿过我的图看了一点，微微的点了点头，嘴角露出思思的笑意，我想你这糟老头子应该不会再刁难我了吧。

面试官： 嗯，你接着演示你的可重复读吧。

我： 嗯，好的，然后就是可重复读，和之前一样的操作。



我： 将两个session开启为 `REPEATABLE READ` ，同时开启事务，在第一个事务中先select，然后在第二个事务里面update数据行，可以发现即使第二个事务已经commit，第一个事务再次select数据也还是没有改变，这就解决了不可重复读的问题。

我： 这里有个不同的地方就是在Mysql中，默认的不可重复读个隔离级别也解决了幻读的问题。

我： 从上面的演示中可以看出第一个事务中先select一个id=3的数据行，这条数据行是不存在的，返回Empty set，然后第二个事务中insert一条id=3的数据行并且commit，第一个事务中再次select，数据也好是没有id=3的数据行。

我： 最后的串行化，样式步骤也是一样的，结果也和Mysql中默认的个可重复读隔离级别的结果一样，串行化的执行流程相当于把事务的执行过程变为顺序执行，我这边就不再做演示了。

我： 这四大等级从上到下，隔离的效果是逐渐增强，但是性能却是越来越差。

面试官： 哦？性能越来越差？为什么会性能越来越差？你能说一说原因吗？

哎呀，我这嘴，少说一句会死啊，这下好了，这个得说底层实现原理了，从原来的假装若有所思，变成了真正得若有所思。

我： 这个得从Mysq的锁说起，在Mysql中的锁可以分为分**享锁/读锁（Shared Locks）**、**排他锁/写锁（Exclusive Locks）** 、**间隙锁**、**行锁（Record Locks）**、**表锁**。

我： 在四个隔离级别中加锁肯定是要消耗性能的，而读未提交是没有加任何锁的，所以对于它来说也就是没有隔离的效果，所以它的性能也是最好的。

我： 对于串行化加的是一把大锁，读的时候加共享锁，不能写，写的时候，加的是排它锁，阻塞其它事务的写入和读取，若是其它的事务长时间不能写入就会直接报超时，所以它的性能也是最差的，对于它来就没有什么并发性可言。

我： 对于读提交和可重复读，他们俩的实现是兼顾解决数据问题，然后又要有一定的并发行，所以在实现上锁机制会比串行化优化很多，提高并发性，所以性能也会比较好。

我： 他们俩的底层实现采用的是MVCC（多版本并发控制）方式进行实现。

面试官： 你能先说一下先这几个锁的概念吗？我不是很懂，说说你的理解。

我： 哦，好的，共享锁是针对同一份数据，多个读操作可以同时进行，简单来说即读加锁，不能写并且可并行读；排他锁针对写操作，假如当前写操作没有完成，那么它会阻断其它的写锁和读锁，即写加锁，其它读写都阻塞 。

我： 而行锁和表锁，是从锁的粒度上进行划分的，行锁锁定当前数据行，锁的粒度小，加锁慢，发生锁冲突的概率小，并发度高，行锁也是MyISAM和InnoDB的区别之一，InnoDB支持行锁并且支持事务 。

我： 而表锁则锁的粒度大，加锁快，开销小，但是锁冲突的概率大，并发度低。

我： 间隙锁则分为两种： `Gap Locks` 和 `Next-Key Locks` 。Gap Locks会锁住两个索引之间的区间，比如select * from User where id>3 and id<5 for update，就会在区间（3，5）之间加上Gap Locks。

我： Next-Key Locks是Gap Locks+Record Locks形成闭区间锁select * from User where id>=3 and id=<5 for update，就会在区间[3,5]之间加上Next-Key Locks。

面试官： 那Mysql中什么时候会加锁呢？

我： 在数据库的增、删、改、查中，只有增、删、改才会加上排它锁，而只是查询并不会加锁，只能通过在select语句后显式加lock in share mode或者for update来加共享锁或者排它锁。

面试官： 你在上面提到MVCC（多版本并发控制），你能说一说原理吗？

我： 在实现MVCC时用到了一致性视图，用于支持读提交和可重复读的实现。

我： 在实现可重复读的隔离级别，只需要在事务开始的时候创建一致性视图，也叫做快照，之后的查询里都共用这个一致性视图，后续的事务对数据的更改是对当前事务是不可见的，这样就实现了可重复读。

我： 而读提交，每一个语句执行前都会重新计算出一个新的视图，这个也是可重复读和读提交在MVCC实现层面上的区别。

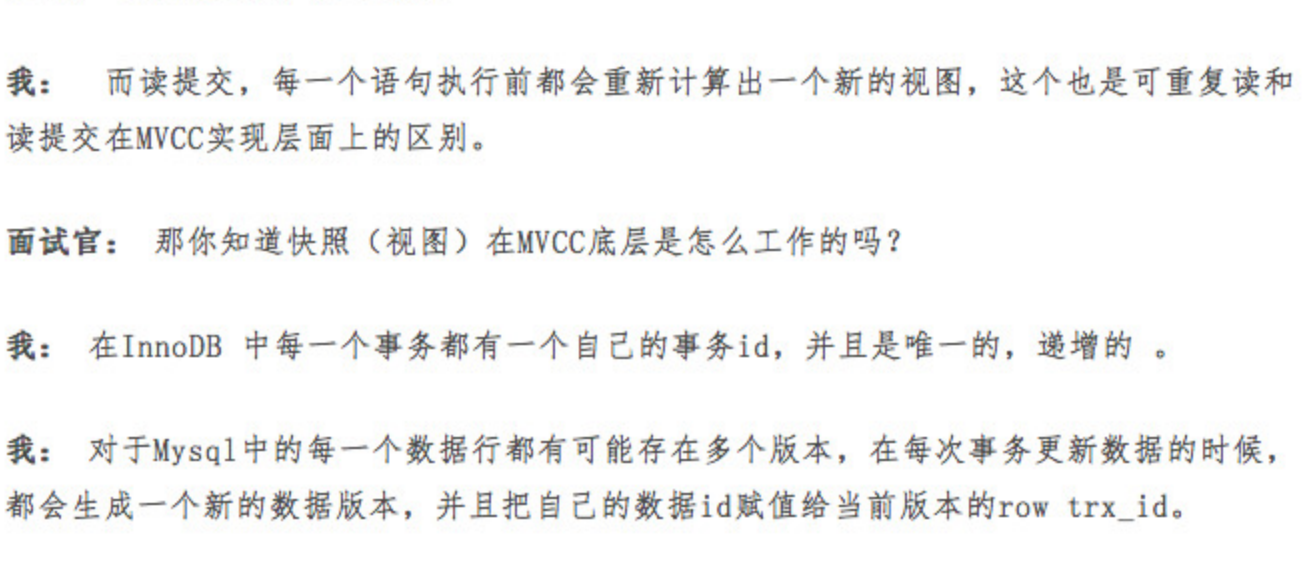
面试官： 那你知道快照（视图）在MVCC底层是怎么工作的吗？

我： 在InnoDB 中每一个事务都有一个自己的事务id，并且是唯一的，递增的 。

我： 对于Mysql中的每一个数据行都有可能存在多个版本，在每次事务更新数据的时候，都会生成一个新的数据版本，并且把自己的数据id赋值给当前版本的row trx_id。

面试官： 小伙子你可以画个图我看看吗？我不是很明白。

我有什么办法呢？完全没办法，只能又从屁股兜里拿出笔和纸，迅速的画了起来，要是这次面试不过就血亏啊，浪费了我两张纸和笔水，现在的笔和纸多贵啊，只能豁出去了。



我： 如图中所示，假如三个事务更新了同一行数据，那么就会有对应的三个数据版本。

我： 实际上版本1、版本2并非实际物理存在的，而图中的U1和U2实际就是undo log，这v1和v2版本是根据当前v3和undo log计算出来的。

面试官： 那对于一个快照来说，你知道它要遵循什么规则吗？

我： 嗯，对于一个事务视图来说除了对自己更新的总是可见，另外还有三种情况：版本未提交的，都是不可见的；版本已经提交，但是是在创建视图之后提交的也是不可见的；版本已经提交，若是在创建视图之前提交的是可见的。

面试官： 假如两个事务执行写操作，又怎么保证并发呢？

我： 假如事务1和事务2都要执行update操作，事务1先update数据行的时候，先回获取行锁，锁定数据，当事务2要进行update操作的时候，也会取获取该数据行的行锁，但是已经被事务1占有，事务2只能wait。

我： 若是事务1长时间没有释放锁，事务2就会出现超时异常 。

面试官： 这个是在update的where后的条件是在有索引的情况下吧？

我： 嗯，是的 。

面试官： 那没有索引的条件下呢？没办法快速定位到数据行呢？

我： 若是没有索引的条件下，就获取所有行，都加上行锁，然后Mysql会再次过滤符合条件的的行并释放锁，只有符合条件的行才会继续持有锁。

我： 这样的性能消耗也会比较大。

面试官： 嗯嗯

此时面试官看看手表一个多钟已经过去了，也已经到了饭点时刻，我想他应该是肚子饿了，不会继续追问吧，两人持续僵了三十秒，他终于开口了。

面试官： 小伙子，现在时间也已经到了饭点了，今天的面试就到此结束吧，你回去等通知吧。

我： 。。。。。。。。。。