3| 隔离性和事务隔离级别

事务就是要保证一组数据库操做，要么全成功，要么全失败，事务是在引擎层实现的

事务的个隔离级别

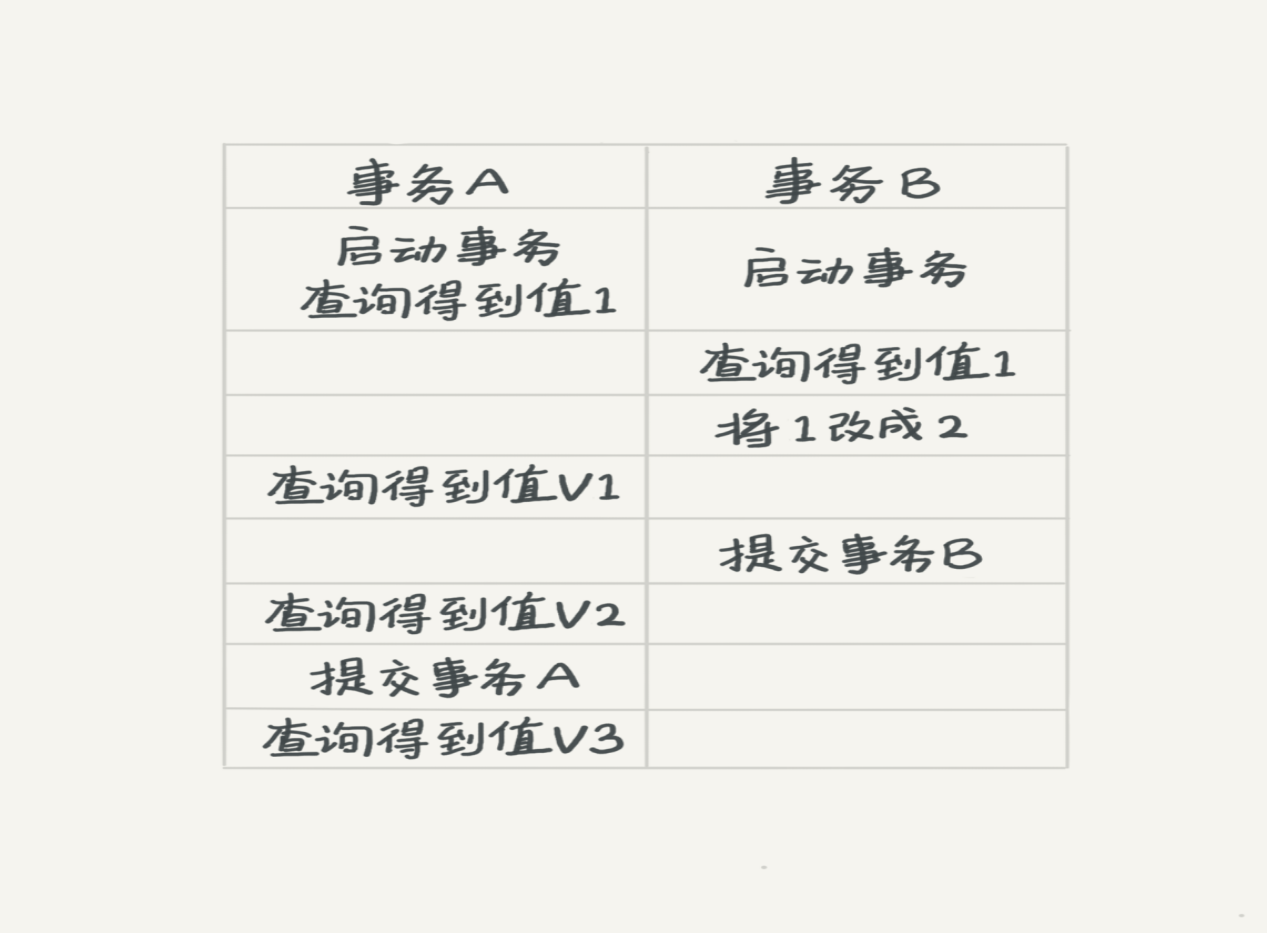
读未提交：一个事务还没有提交时，他做的变更就能被别的事务看到

读已提交：一个事务提交后，它做的变更才会被其他事务看到

可重复读：一个事务执行过程中看到的数据，总是跟这个事务在启动是看到的数据是一直的。当然在可重复读隔离级别下，未提交变更对其他事务不可见

串行化：对于同一行记录，“写”会加“写锁”，“读”会加“读锁”，当出现读写锁冲突

后访问的事务必须等前一个事务执行完成，才能继续。



在不同的隔离级别下，A会返回那些不同的结果，V1,V2,V3会返回那些值

1)读未提交：v1 是2 ，此时事务B还没有提交，但结果已经被A看到，因此v2是2，v3是2，

2)读已提交：v1 是1 ，v2,是2，事务B的更新在提交后才能被A看到，所以V3的值是2

3)可重复读：v1 是1 ，v2是 1，v3 是2，之所以V2是1，遵循的就是这个要求：事务在执行期间看到的数据前后必须一致

1. 串行化：事务B在执行将1改为2时，会被锁住，知道事务A提交后，事务B才可以继续执行，所以A ，V1,V2是1， V3是2

实现上，数据库里面会创建一个视图，访问的时候以视图的逻辑结果为准

1. 读未提交：直接返回记录上的最新值，没有视图概念

2.读已提交：这个视图是**每个SQL语句开始执行的时候创建的**

3.可重复读：这个视图是**事务启动时创建的**，整个事务存在期间都用这个视图

4.串行化：直接加锁的方式来避免并行访问

将启动参数 tx\_isolation值设置为READ-COMMITED

Show variables来看当前的值

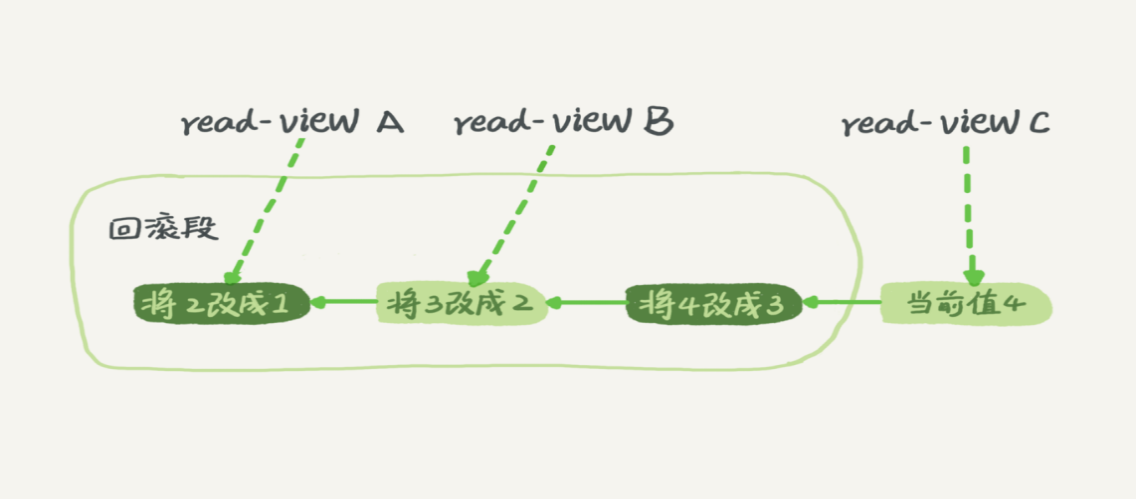
Shou variables like ‘tx\_isolation’;

**事务隔离级别的实现 需要补充**

可重复读

实际上每条记录更新的时候都会同时记录了一条回滚操作，记录上的最新值，通过回滚操作，都可以到前一个状态的值

假设 一个值从1按顺序该成2,3,4,在回滚日志里就有类似的下面的记录



当前值是4，但是查询这条记录的时候，不同时刻启动的事务会有不同的read-view,如图中的，再视图A,B,C 这个记录的值分别是1,2,4,同一条记录在系统中可以存在多个版本，就是数据库的多版本并发控制（MVCC）,对于视图read-viewA要得到1,就必须将当前值依次执行图中所有的回滚操作得到。

即使现在另外一个事务将4改成5，这个跟read-viewA,B,C对应的事务不会冲突

你一定会问，回滚日志总部不能一直保留吧，什么时候删除呢？

不需要的时候删除，系统会判断，当前没有事务再需要用到这些回滚日志时，回滚日志会删除。什么时候不需要它了，就是当系统里没有比这个回滚日这更早的read-view的时候。

基于上面的说明，我们来讨论一下为什么建议尽量不要是用长事务？

长事务意味着系统里面会存在很老的事务视图，由于这些事务随时可能访问数据库里面的任何数据，所以事务提交前，数据库里面它可能用到回滚记录都必须保留，这就会导致大量占用存储空间。除了对回滚段的影响，长事务还占用锁资源，也可能拖垮整个库

事务的启用方式

1. 显式启动事务语句，begin或start transaction。配套的提交语句是commit,回滚语句是rollback.
2. set autocoommit = 0,整个命令会将整个线程的自动提交关掉，意味着只执行一个select语句，这个事务就启动了，并且不会自动提交，这个事务持续存在直到主动执行commit或rollback语句，或断开连接。

有些客户端连接框架会默认连接成功后先执行一个set autocommit=0的命令，这导致接下来的查询都在事务中，如果是长连接，导致意外的长事务。

因此，建议总是使用set autocommit=1 通过显示语句的方式来启动事务

有些同学纠结“多一次交互的问题”，对于一个需要频繁使用事务的业务，第二种方式每个事务在开始是都不需要主动执行一次“begin”,减少语句的交互次数。如果有这个顾虑，建议使用commit work and chain 语法

在autocommit为1 的情况下，用begin显示启动的事务，如果执行commit则提交事务。如果执行commit work and chain.则是提交事务并自动开启下一个事务，这样也省去了再次执行begin的开销，同时带来的好处是从程序开发的角度明确的知道了每个语句是否处于事务中。

可以在information\_schema库的innodb\_trx这个表中查询长事务，

用于查询持续时间大于60秒的事务

select \* from information\_schema.innodb\_trx where TIME\_TO\_SEC(timediff(now(),trx\_started))>60

**讨论题**

现在知道系统里应该避免长事务，如果你是业务开发负责人同时也是数据库负责人，你会有什么办法来避免出现或者处理这种情况？

**首先，从应用开发端来看**

1. 确认是否使用了set autocommit= 0 ，这个确认工作可以在测试环境中展开，把MySQL的general\_log开起来，然后随便跑一个业务逻辑，通过general\_log的日志来确认，一般框架如果会设置这个值，也就会提供参数来控制行为，你的目标就是把它改成1；
2. 确认是否有必要的只读事务，有些框架会习惯不管什么语句先用begin/commit框起来，我见过有些事务没有这个必要，但是也把好几个事务放到了事务中，这种只读事务可以去掉。
3. 业务连接数据库时，根据业务本身的预估，通过SET MAX\_EXECUTION\_TIME命令，来控制每个语句执行的最长时间，避免每个语句意外执行太长时间（为什么会意外，后续章节提到这类案例）

**其次，从数据端看**

1. 监控information\_schema.innodb\_trx表，设置长事务阈值，超过就报警/或 kill;
2. Percona 的pt-kill这个工具不错，推荐使用；
3. 在业务功能测试阶段要求输出有的peneral\_log,分析日志行为提前发现问题
4. 如果使用的MySQL5.6或者跟新版本，把innodb\_undo\_tablespaces设置为2(或更大的值）若果真的出现大事务导致回滚过大，这样设置清理起来更方便。