链滴

事务隔离

作者：xhaoxiong

原文链接：https://ld246.com/article/1637547905911

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

<h4 id="什么是事务-">什么是事务？</h4>
简单来说，事务就是要保证一组数据库操作，要么全部成功，要么全部失败。在 MySQL 中，事支持是在引擎层实现的。并不是所有的引擎都支持事务。比如 MySQL 原生的 MyISAM 引擎就不支事务，这也是 MyISAM 被 InnoDB 取代的重要原因之一。
<h4 id="什么是隔离性与隔离级别-">什么是隔离性与隔离级别？</h4>
当数据库上有多个事务同时执行的时候，就可能出现脏读（dirty read）、不可重复读（non-repatable read）、幻读（phantom read）的问题，为了解决这些问题，就有了“隔离级别”的概念。
隔离得越严实，效率就会越低。因此很多时候，我们都要在二者之间寻找一个平衡点。SQL 标准事务隔离级别包括：读未提交（read uncommitted）、读提交（read committed）、可重复读（reeatable read）和串行化（serializable ）。
读未提交是指，一个事务还没提交时，它做的变更就能被别的事务看到。
读提交是指，一个事务提交之后，它做的变更才会被其他事务看到。
可重复读是指，一个事务执行过程中看到的数据，总是跟这个事务在启动时看到的数据是一致的当然在可重复读隔离级别下，未提交变更对其他事务也是不可见的。
串行化，顾名思义是对于同一行记录，“写”会加“写锁”，“读”会加“读锁”。当出现读写冲突的时候，后访问的事务必须等前一个事务执行完成，才能继续执行。
<h5 id="例子-">例子：</h5>
mysql> create table T(c int) engine=InnoDB;
insert into T(c) values(1);

/b3logfile.com/file/2021/11/image-e08531da.png?imageView2/2/interlace/1/format/jpg"></
>
<h6 id="情况1-隔离级别为-读未提交--则v1-2-v2-2-v3-2-未提交的事务A是能够读取到事务B过程的值-">情况 1：隔离级别为“读未提交”，则 v1=2，v2=2，v3=2。未提交的事务 A 是能够读取事务 B 过程中的值。</h6>
<h6 id="情况2-隔离级别为-读提交--则v1-1-v2-2-v3-2-未提交事务A是能够读取到事务B提交事务的值-">情况 2：隔离级别为“读提交”，则 v1=1，v2=2，v3=2。未提交事务 A 是能够读取到事务 提交事务后的值。</h6>
<h6 id="情况3-隔离级别为-可重复读--则v1-1-v2-1-v3-2-已提交的事务A才能够读取到事务B提交务后的值-也就是说遵循-事务在执行期间看到的数据前后必须是一致--的-">情况 3：隔离级别为“可复读”，则 v1=1，v2=1，v3=2。已提交的事务 A 才能够读取到事务 B 提交事务后的值。也就是说循*事务在执行期间看到的数据前后必须是一致* 的。</h6>
<h6 id="情况4-隔离级别为-串行化--则v1-1-v2-1-v3-2-在事务-B-执行-将-1-改成-2-的时候-会被住--直到事务-A-提交后-事务-B-才可以继续执行--">情况 4：隔离级别为“串行化”，则 v1=1，v21，v3=2。在事务 B 执行“将 1 改成 2”的时候，会被锁住。*直到事务 A 提交后，事务 B 才可以继执行。*</h6>
在不同隔离级别的情况下，数据库执行会有所不同。因此对于一些从 Oracle 迁移到 MySQL 的用，为保证数据库隔离级别的一致，一定要记得将 MySQL 的隔离级别设置为“读提交”。
配置的方式是，将启动参数 transaction-isolation 的值设置成 READ-COMMITTED。你可以用how variables 来查看当前的值。
mysql> show variables like 'transaction_isolation';
+-----------------------+----------------+
| Variable_name | Value |
+-----------------------+----------------+
| transaction_isolation | READ-COMMITTED |
+-----------------------+----------------+
<h6 id="可重复读可能发生在什么场景呢-">可重复读可能发生在什么场景呢？</h6>
假设你在管理一个个人银行账户表。一个表存了账户余额，一个表存了账单明细。到了月底你要数据校对，也就是判断上个月的余额和当前余额的差额，是否与本月的账单明细一致。你一定希望在对过程中，即使有用户发生了一笔新的交易，也不影响你的校对结果。
这时候使用“可重复读”隔离级别就很方便。事务启动时的视图可以认为是静态的，不受其他事更新的影响。
<h5 id="事务隔离的实现">事务隔离的实现</h5>

在 MySQL 中，实际上每条记录在更新的时候都会同时记录一条回滚操作。记录上的最新值，通
回滚操作，都可以得到前一个状态的值。假设一个值从 1 被按顺序改成了 2、3、4，在回滚日志里面
会有类似下面的记录。

/b3logfile.com/file/2021/11/image-ad1963a9.png?imageView2/2/interlace/1/format/jpg"></
>
当前值是 4，但是在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
到的，在视图 A、B、C 里面，这一个记录的值分别是 1、2、4，同一条记录在系统中可以存在多个
本，就是数据库的多版本并发控制（MVCC）。对于 read-view A，要得到 1，就必须将当前值依次
行图中所有的回滚操作得到。
同时你会发现，即使现在有另外一个事务正在将 4 改成 5，这个事务跟 read-view A、B、C 对
的事务是不会冲突的。
回滚日志总不能一直保留吧，什么时候删除呢？答案是，在不需要的时候才删除。也就是说，系
会判断，当没有事务再需要用到这些回滚日志时，回滚日志会被删除。
什么时候才不需要了呢？就是当系统里没有比这个回滚日志更早的 read-view 的时候。
<h5 id="为什么尽量不要使用长事务-">为什么尽量不要使用长事务？</h5>
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
，所以这个事务提交之前，数据库里面它可能用到的回滚记录都必须保留，这就会导致大量占用存储
间。在 MySQL 5.5 及以前的版本，回滚日志是跟数据字典一起放在 ibdata 文件里的，即使长事务最
提交，回滚段被清理，文件也不会变小。
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。
<h5 id="事务的启动方式">事务的启动方式</h5>
MySQL 的事务启动方式有以下几种：

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 rollb
ck。
set autocommit=0，这个命令会将这个线程的自动提交关掉。意味着如果你只执行一个 select
句，这个事务就启动了，而且并不会自动提交。这个事务持续存在直到你主动执行 commit 或 rollbac
语句，或者断开连接。

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。这就导致接下
的查询都在事务中，如果是长连接，就导致了意外的长事务。
因此，最好使用 set autocommit=1, 通过显式语句的方式来启动事务。
但是有的开发同学会纠结“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
每个事务在开始时都不需要主动执行一次 “begin”，减少了语句的交互次数。最好使用 c
mmit work and chain 语法。
在 autocommit 为 1 的情况下，用 begin 显式启动的事务，如果执行 commit 则提交事务。如
执行 commit work and chain，则是提交事务并自动启动下一个事务，这样也省去了再次执行 begin
语句的开销。同时带来的好处是从程序开发的角度明确地知道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，比如下面这个语句，用于
找持续时间超过 60s 的事务。
select * from information_schema.innodb_trx where TIME_TO_SEC(timediff(now(),trx_start
d))>60