zhuanlan.zhihu.com /p/433231172

事务之ACID

随着科技的飞速发展，人类社会也迈入了大数据时代。很多数据的值不准对我们的生活影响不会很大，比如手表记录下来的我今天走路步数是10000步，实际就算记成了9500步对我来讲也不会太关心。但是有些数据就分毫不能差，比如银行卡里的钱不能莫名其妙的就少了500块钱。所有的数据肯定都需要存在在一些数据系统里面，比如数据库，硬盘，云存储等等。但是现在的应用越来越复杂，在访问数据系统经常出现各种问题，比如：

- 写数据到数据库的时候出现了软件或硬件故障导致数据写了一半失败了；
- 应用突然崩了；
- 网络突然断了导致应用连接不上数据库了；
- 多个客户端想要同时更新同一个值导致后者覆盖了前者。

比如小明想给小华转账100块钱，银行应用的业务逻辑是如果小明账号里的余额大于100元时，会从小明的账号里减去100元，然后把小华的账号余额加上100元。但是如果刚从小明的账号里扣除100元时，银行网断了导致后面给小华的钱没加成功，这样就出现了重大问题，在银行转账时绝对不能发生。所以银行的应用开发时需要考虑这种异常并进行处理，比如等来网了的时候把小华的账号里加上100块钱。

我们可以在应用端处理各种异常，让应用变得更加鲁棒，但是其中这里面有很多细节的问题需要考虑，非常麻烦。而事务（Transaction）是一种可以简化问题的机制。

1. 什么是事务

事务可以把一个应用的多个读写操作合并为一个逻辑单元。理论来说，一个事务中的所有读写操作执行的时候就像是一个操作一样：或者整个事务的所有操作成功，或者整个事务的所有操作全部失败。 不允许出现在一个事务中：其中的一部分操作成功了，但是另一部分的操作失败了的情况。这样就简化了问题。比如上面的例子中，把扣除小明账号的钱和增加小华账号里的钱看作一个事务，那中间网断了最多就是没转帐成功，两边的钱都没变化，这就保证了不会存在钱转丢了的情况。这样也方便应用进行重试（Safely Retry）。

大家可能之前都听说过或用到过事务，看上去好像数据库就应该使用事务。但是事务并不是本来就天然存在的，他是为了简化访问数据库时的编程模型而被创造出来的概念。使用事务可以帮助开发者忽略一些潜在的数据问题和并发问题，因为实现事务的数据库本身会帮忙处理这类问题。

那既然事务这么好，是不是所有数据库都实现了事务，我们就直接用就行，不需要了解事务的具体原理了。实际不是的。因为事务的实现有很多种方式以及不同的级别，他们对应用性能的影响也是不同的。比如一种事务隔离性的实现方式就是加锁，我们把一个事务里涉及的所有数据行都加上锁，这样别的事务根本不能读写我们锁下的这些行数据，直到事务结束才释放这些锁，这样肯定就能避免数据不一致的情况了。但是这样相当于把读写数据串行化了，会非常影响性能。银行数据非常重要，这样实现虽然慢但是保证数据的安全了也还能接受。但是假设我们的应用中数据没有那么重要的情况下，可能我们这种拿性能换数据一致的做法就不太合理了。因此**不是每个应用都需要事务。**

尽管事务看上去简单直接，但是实际有很多细节需要考虑情况，我们将会一一进行介绍。首先先介绍一下数据库中 ACID 的概念。

2. 什么是 ACID

事务提供的安全保证 (Safety Guarantee)可以用 ACID 来描述分别是：

- 原子性 (Atomicity)
- 一致性 (Consistency)
- 隔离性 (Isolation)
- 持久性 (Durability)。

这是一些事务的安全保证，但是不同数据库对于ACID的实现可能也是不同的。比如对于隔离性就存在巨大的歧义。所以今天一个数据库说自己满足ACID要求，但是可能和你以为的ACID并不一样。下面分别对这四种安全保证进行解释。

2.1. 原子性 (Atomicity)

原子这个词很容易造成误解。大家很容易联想到多线程中的原子操作。如果一个线程执行原子操作，表示其他线程不能看到这个原子操作的中间结果。

但是在 ACID 中，原子性和并发无关，也就是说 ACID 中的原子性不表示当多个进程想要在同一时刻访问同一数据时会发生什么，但是隔离性表示的是这个意思。

ACID 中的原子性描述了如果一个事务中间出现了错误（比如网络突然断了）导致这个事务不可能完成，那么事务必须回滚到事务开始之前的状态。 也就是说删掉这个事务已经写到数据库中的修改。

原子性保证了事务中的操作要么全都发生，要么全都不发生，不可能一半完成了，一半没做。其实回滚性 (Abortability) 这个名字比原子性更能表示这个特征，但是原子性还是更通用的一个叫法。

举例来说，小明要给小华转账100元。事务里包含两个操作，第一个是从小明的账号里扣除100元，第二个是在小华的账号里增加100元。原子性保证这个事务要么操作全成功，要么操作全失败，不能出现小明的账户里少了100元，小华的账号里钱数没变。

2.2. 一致性 (Consistency)

ACID 中的一致性限制了我们自定义的一些数据约束永远为真。 比如上面银行转账的例子，银行定义的一个约束条件是无论怎么转账，大家的存款总额不变。

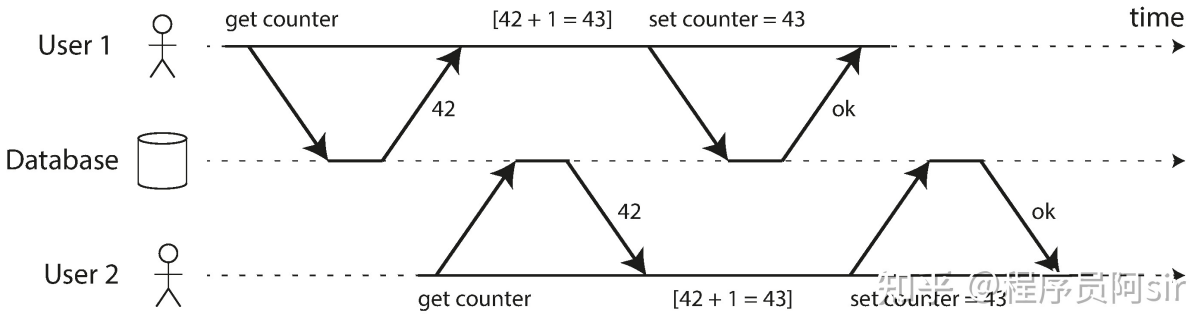
但是一致性实际取决于应用本身的定义，也就是说应用负责定义哪些东西需要保持一致性。比如应用逻辑就是要求扣小明100然后给小华账号加200块，数据库也不能阻止他这样做，因为这个是应用里面定义的。数据库可能可以加一些外键约束或者唯一性约束，但是一般来讲，应用负责定义什么数据是合理的，什么是不合理的，数据库只负责存储。

有意思的一点是：原子性、隔离性、持久性都是数据库的性质，而一致性是应用的性质。应用需要依数据库的原子性和隔离性来实现一致性。所以从某种意义来说，ACID 中的 C 不应该属于数据库的安全性保证范畴。

2.3. 隔离性

一个数据库可以被多个客户端访问，如果他们想要读写数据库的不同部分肯定没有问题，但是如果他们想要同时访问数据库的同一条记录，就有可能产生并发问题（Concurrency Problem)，也叫竞争条件 (Race Conditions)。

比如下面的例子：



用户1和用户2都想增加数据库中 counter 的值。在修改之前是42。用户1读到值为42，在用户1改变 counter 的值之前，用户2也读到了当前值为42，然后用户1和用户2分别在原始值上加1，得到43，然后写入了数据库。最终数据库中 counter 的值变成了43，但实际正确的值应该是44。

隔离性要解决的就是这个并发问题。**隔离性意思是并发执行的事务之前彼此相互隔离，不应该相互影响。**

隔离性是 ACID 中最复杂的，也存在着很多概念上的争议。大家可能认为隔离性就是可串行化 (Serializability)。意思是数据库需要保证这些事务在并发情况下运行最后的结果需要和串行运行这些事务的结果一致。实际上这确实是一种隔离性的实现方案，也是最高的隔离级别。但是这种实现却很少会被用到，因为这会导致性能严重受到影响。有些数据库甚至根本没有实现这种方案，比如Oracle。Oracle 中有一种隔离级别是“可串行的” (Serializable)，但是他实现的是一种较弱的隔离级别--快照隔离 (Snapshot Isolation)，有种挂羊头卖狗肉的感觉... 各种隔离级别的区分我们会在下一篇文章中介绍。

2.4. 持久性

持久性承诺一旦一个事务被成功完成，所有的数据改动将不会回滚。即使硬盘坏了也能保证数据可以被持久存储。

实际上没有任何技术可以保证数据绝对持久保存。大家都只是用一些减少数据丢失的技术，比如写到硬盘、备份等等。

总结

这篇文章介绍了什么是事务以及数据库安全保证中的 ACID 的含义。下一篇文章将继续介绍事务的更多细节。

参考文献

[1] Kleppmann, Martin. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. " O'Reilly Media, Inc.", 2017.