Isolation Level

22 Apr 2015 by fleuria

很久之前就想记一下关于数据库隔离级别的东西了，前几周在知乎上回答了一个相关的问题，正好整理一下。

完美的数据正确性有它的代价，不同的读写场景，对隔离性的需求不同。隔离性越高，数据越安全，但性能越低。教科书上一般会写四种隔离级别，按照不同隔离级别可能出现的 “症状” 划分：

- Read Uncommitted：可能读到他人未 Commit 、可能被 Rollback 的脏数据，即 Dirty Read；
- Read Committed：不会读到他人未 Commit 的数据，但事务内重复读一条数据，可能得到不同的结果，即 Non-repeatable Read；
- Repeatable Read：同一事务内重复读一条数据的结果相同，但读取多个数据的时候，可能会得到不同的结果，即 Phantom Read；
- Serializable：完全的隔离性，在语义上相当于事务的执行没有并发一样；

时至今日即使 wikipedia 上也是这四种分类，但是与现实世界已经不符合了[1]：按照 “症状” 划分的四种隔离级别，只适用于描述基于锁实现的并发控制；可是现在市面上的数据库早就都基于 MVCC 实现了，结果是各家数据库依然留着这四个隔离级别的名头，但语义有了不同：

- MySQL 中根本不会出现 Phantom Read；Repeatable Read 其实是 Snapshot Isolation ；只有 Serializable 是完全基于锁的；
- Postgres 实质上只有两种隔离级别：Read Committed 和 Serializable；而 Serializable 是基于 MVCC 的无锁实现，即 Serializable Snapshot，相比 Snapshot 只牺牲一点性能，但彻底杜绝了出错的可能性；

基于 MVCC 的 Snapshot 隔离级别看起来很美，在开始事务的一刻拍下快照，不会出现 Non-repeatable read 和 Phantom Read；但是它也有它的 “症状” 而导致丢失写，即 Write Skew。大约来看它的性质是：

- 对一张表的多行数据的并发修改满足隔离性，一旦存在冲突的写入，会触发回滚；
- 但是如果涉及以多张表返回的结果计算出新结果，写入另一张表，则容易出现 Write Skew；

Write Skew 的一个例子是：

- 假设银行系统中的一个用户有两个账户：支票账户和储蓄账户，对应两张表；
- 假设存在不变量，要求用户两个账户的余额总额总是大于 1000。
- 如果存在两个事务分别从两个账户中取 100，在提交时不会发生冲突，却会破坏我们的不变量；

所以不能期望加了一个事务就万事大吉，而要了解每种隔离级别的语义。涉及单行数据事务的话，只要 Read Committed ＋ 乐观锁就足够保证不丢写；涉及多行数据的事务的话，Serializable 隔离环境的话不会出错，但是你不会开；如果开 Repeatable Read （Snapshot）隔离级别，那么可能会因为 Write Skew 而丢掉写。

更糟糕的是，Write Skew 如果存在写冲突，不会触发 Rollback，而只是默默地把数据弄坏。这时，就要求我们在读取来自两张表的数据时，手工地通过 SELECT .. FOR UPDATE 上锁。

你会发现，Snapshot 隔离级别的语义和坑反而比较难清晰地描述和规避。既然出错是默默地出错，手工上锁没准哪里就上漏了，你怎么向老板拍胸脯你的事务没有错误呢？

Postgres 意识到 Snapshot 隔离级别反而更容易出错，但是针对 Snapshot 隔离级别的特定几个错误场景做检查的话，即可以较低的开销保证正确性。它的 Repeatable Read 级别，其实就做到 Serializable 了，可以放心使用。

到最后还是跟业务场景相关，毕竟不同的业务场景，对正确性的需求并不相同：

- 如果是互联网业务，对一致性的要求往往没有太高，一般不需要特别高的隔离级别；与其在意隔离级别，可能会更多在意主从不同步造成的脏数据和缓存的不一致；但是务必手工 Commit，在 Commit 成功之后再加缓存；务必将写操作套在事务里，并看需求适当地上锁；单行操作的话，优先使用乐观锁，最好使用一个有内建乐观锁支持的 ORM，可以避免走漏；
- 如果是金融业务，尽量不要用 MySQL 好了， Postgres 的 Serializable Snapshot 太良心，不需要手工锁；除了先验地避免并发错误，也需要一些后验的手段，去发现并发错误，比如对于关键的数据操作，记录下所有的变更，定期做对账，看看账平不平，即使是金融系统，100% 的数据正确性也不一定是绝对的，尽力保证正确性，并留有查账修账的手段。

此外，数据库的各种约束（比如唯一性约束）可以保证在各种隔离级别下都保持一致，在早期对事务安全性不够信心时，可以先多加一些约束，如果线上跑一段时间没有异常，再放松一些约束。

- 1: http://www.cs.umb.edu/~poneil/iso.pdf
- 2: PostgreSQL: Documentation: 8.0: Transaction Isolation