zhuanlan.zhihu.com /p/437847549

事务的弱隔离级别之快照隔离

上一篇文章介绍了第一种**弱隔离级别**，也是最常用的一种：**Read Committed**。

程序员阿sir：事务的弱隔离级别之Read Committed9 赞同 · 0 评论文章

但是它会导致的一个并发问题是**不可重复读**。

针对这个并发问题，本篇文章介绍另一种弱隔离级别来解决这个并发问题——**快照隔离**。

2. 快照隔离 (Snapshot Isolation)

很多数据库对快照隔离的命名不同，比如：

- **Oracle** 称之为：可序列化 (Serializable)。
- **PostgreSQL** 和 **MySQL** 称之为：可重复读 (Repeatable Read) 或 快照隔离 (Snapshot Isolation)

2.1. 概念

快照隔离的核心思想是：

> 每个事务都从一个**数据库的快照**中读数据。
> （即：事务看到的所有数据，都是在事务**开始的时间点之前 committed** 的数据。）
> 如果有些数据在当前事务开始之后，被其他事务改变了值，快照隔离能够保证当前事务**无法看到这个新值**。

从思想上看，快照隔离因为每一次读都是从一个**过去的快照**中读取的，不会出现多次读一个值却读到不一致结果的情况。

因此，能完美的解决“不可重复读”的并发问题。

2.2. 如何实现快照隔离

考虑到**性能**问题，一个关键的原则是:

> **读操作**不能阻塞**写操作**，**写操作**也不能阻塞**读操作**，**读操作**也不能阻塞**其他读操作**。

因为**不同的事务**可能需要读取**不同时间点**的数据库快照，所以数据库必须对每一个对象都**维护多个不同的版本**。

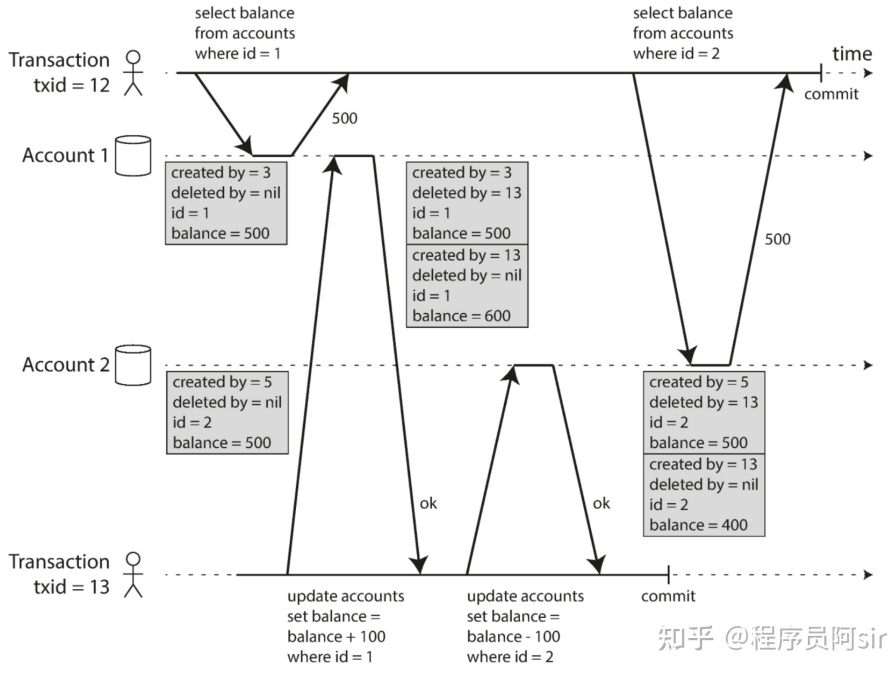
这个技术称为**多版本并发控制 (Multi-Version Concurrency Control (MVCC))**。

如果只需要实现**Read Committed 隔离**，那只需要对每个对象维护两个版本：

- 一是已经 committed 的版本。
- 二是修改了但未 committed的版本。

但是，对于**快照隔离**，就需要维护**多个**版本。

下图说明了 **PostgreSQL** 是如何实现**基于MVCC的快照隔离**的：



基于MVCC的快照隔离实现

当一个事务开始时，数据库会给这个事务分配一个**自增的、独一无二**的**事务ID (Transaction ID)** `txid`。
当这个事务**修改**了某个数据时，这个数据里面就会**标记**上这个事务的ID。
上图所示的每一行数据包含一些内容：

- `created_by`：表示这个数据是被哪个事务加到数据库的。
- `deleted_by`：表示删除这个数据的事务ID。初始值是空。

当一个事务删除或修改一个数据时，实际原来的数据**并没有被删除**，而是把原来的数据的 `deleted_by` 字段标记成了**当前事务的ID**，然后加了一个以**当前事务ID**为 `created_by`的**新数据**。
【删除操作】：当数据库确认旧的数据已经不会被任何事务访问的时候，数据库的**垃圾回收进程**就会释放多余的空间。
【更新操作】：可以理解为会在数据库内部被转换成**先删除、再创建**的流程。

2.3. 快照隔离的问题

有一些并发问题快照隔离也**无法避免**。

比如 **写时序异常 (Write Skew)** 或 **幻读 (Phantoms)**：

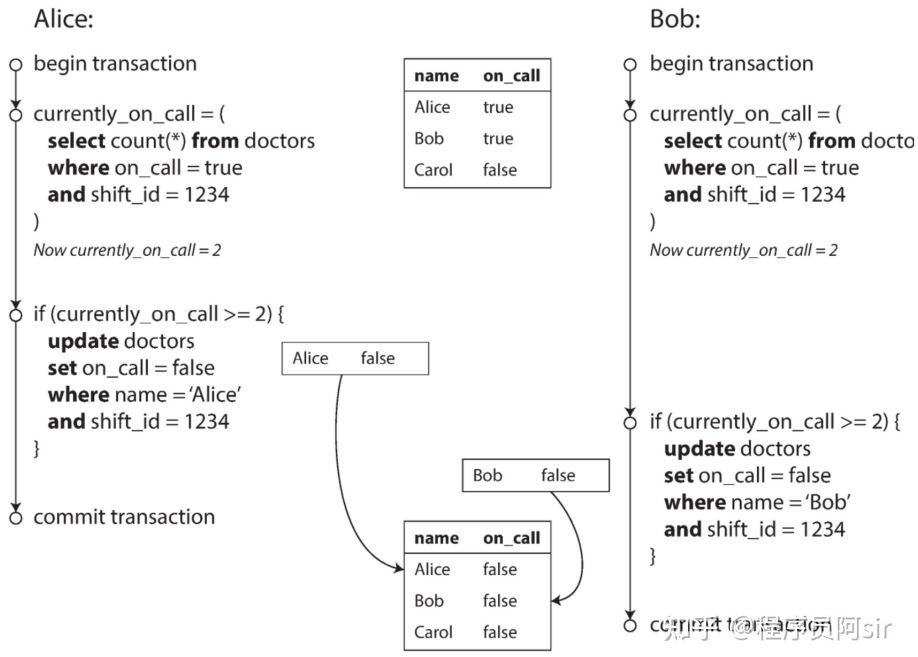
当两个事务读同一个对象，然后同时想要更新不同的对象时，可能出现**写时序异常 (Write Skew)**。

注意：这时两个事务写的是不同的对象，因此不算脏写。可以称为**幻读 (Phantoms)**。

（【幻读】：在一个事务中的写操作会改变另一个事务中读操作的结果。）

举例来说：

假设：医院医生需要**值班** (On-Call)，医院可能安排几个医生在**同一时间**值班，但是至少需要保证**每个时间至少有一名医生**值班。
有 Alice 和 Bob 两个医生，本来被安排在**同一时间**值班。
但是，他们今天都身体不舒服，想要**请假**。
更加不巧地是，他们在**同一时间**点了请假的按钮，导致发生了下面的流程：



Write Skew导致并发问题的例子

每个事务都是先检查当前值班的医生**数量**，如果自己**不是唯一**的一个，就把自己设置为今天**不值班**。
但是这里由于是**快照隔离**，因此，读某个时间值班人数的返回值都是 `2`，即两个事务都能继续往下进行：
Alice把她自己的记录更新成 `on_call = false`。
Bob也把他自己的记录更新成 `on_call = false`。
结果：当前时间便没有人值班了。

一种可行的解决方法是**加锁**。

比如上面的例子：

把 `SELECT` 查询涉及到的所有行**加锁**，这样当其他事务想要**更新**时都需要**等待锁被释放**。

但是，大多数情况我们不知道如何加锁。

这时，可以考虑另一种隔离级别：

可串行化隔离 (Serializability)

我们将在**下一篇**文章中介绍它。