PostgreSQL 中的 MVCC

概述

为什么需要 MVCC?

- 并发读写
  - 在数据的并发读写过程中，由于写入并不是原子性的，因此当一个线程正在写时，如果另一个线程进行读操作的话就很有可能产生数据不一致的问题
  - 比如数据的前半部分写入了，但是后半部分尚未写入，那么在读取时就会取到中间值，也就是脏数据，典型案例就是 64 位整型的写入将会分为两次写入
- 虽然可以使用读写锁来解决并发读写的问题，但是会导致数据库的并发性能降低，并且对于绝大多数应用而言，都是读多写少的，每一个更新操作都会阻塞读取操作，这是我们不能接受的
- 并发控制手段
  - ❶ 悲观并发控制 —— 即严格的两阶段锁（S2PL），语句执行时获取锁，事务结束后释放锁
  - ❷ 乐观并发控制 —— 尝试进行更新，若更新失败则重试
  - ❸ 多版本并发控制（MVCC） —— 每一个写操作都会创建一个新版本的数据项，并保留其旧版本，当事务读取对象时选择一个合适的版本，已确保各个事务之间隔离性的正确性

实现方式

- ❶ 写入数据时将旧数据迁移到另一个地方 —— 例如 MySQL 中的回滚段（undo log），其他线程在读取改行数据时，从回滚段中将旧数据读出来
- ❷ 直接将新数据插入到相关表页中 —— 在同一个存储区域中保存数据的多个版本

Tuple 结构

事务 ID

- PostgreSQL 使用了一个 32 位无符号自增整数来作为事务标识以比较新旧程度
- 可以通过 `txid_current()` 函数来获取当前事务的标识 —— select txid_current();

HeapTupleHeaderData

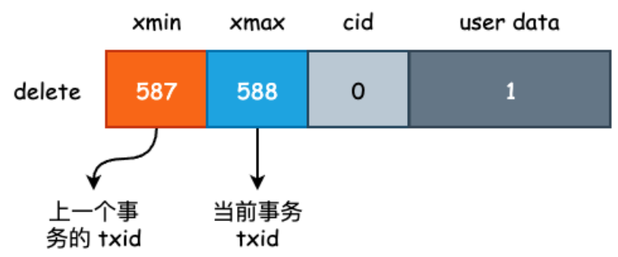
- 上图为一条数据（元组）在磁盘中的实际存储结构，与 MVCC 相关的字段主要包括 4 个
- t_xmin —— 保存了插入该元组的事务的 txid
- t_xmax —— 保存删除或者是更新该元组的事务的 txid，若一个 tuple 既没有被更新也没有被删除的话，该字段的值为 0
- t_cid —— 即 Command ID，表示在当前事务中，执行当前命令之前共执行了多少条命令，从 0 开始计数。t_cid 的主要作用就在于判断游标的数据可见性
- t_infomask —— 位掩码，主要保存了事务执行的状态，如 XMIN_COMMITTED、XMAX_COMMITTED 等。同时也保存了 COMBOCID 这一非常重要的标识位，也是和游标相关的字段

插入

- t_xmin 与创建相关，当我们 insert 一条数据时，t_xmin 就会被设置成执行事务的 txid，并且一旦设置，便不会修改

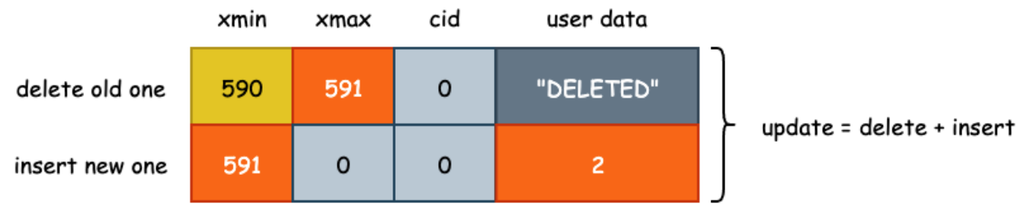
删除

- t_xmax 与删除有关，当我们删除一条数据时，t_xmax 就会被设置成执行事务的 txid



更新

在 PostgresSQL 中，元组的更新并不是原地的，也就是说新数据不会覆盖旧有的数据，而是通过将旧数据标记为删除，新插入一条数据的方式来完成更新。也就是说，假如说我们对 100 条数据进行更新的话，最终会在文件中产生 200 条数据，其中有 100 条被标记为删除

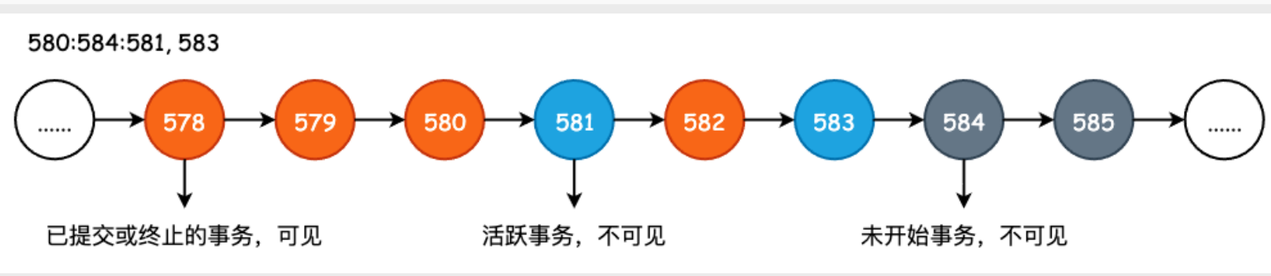


事务快照与基本可见性判断

事务快照

事务快照是一个数据集合，保存了某个事务在某个特定时间点所看到的事务状态信息，包括哪些事务已经结束，哪些事务正在进行，以及哪些事务还未开始，我们可以通过 txid_current_snapshot() 函数来获取当前的事务快照

txid_current_snapshot() 的文本表示含义为 xmin:xmax:xip_list，其中 xmin 表示所有小于它的事务要么已提交，要么已经回滚，即事务结束。xmax 则表示第一个尚未分配的 txid，即所有 txid >= xmax 的事务都还没有开始。而 xip_list 则是使用逗号分割的一组 txid，表示在获取快照时还是进行的事务



基本可见性判断（排除游标）

可参考: https://smartkeyerror.com/PostgreSQL-MVCC-01