12 并发控制

- 要点
  - 2PL(two-phase locking) design 两相加锁协议
  - deadlock
- 基于锁的协议
  - 锁
    - 共享的：可读不可写，任意数量的事务都可以持有一个项目的共享锁
    - 排他的：既可读又可以写
    - 相容的：对数据项申请加A类型锁，但此时该数据项上有B类型锁，但A可以立即获得
  - 死锁
    - 彼此申请对方拥有的锁，事务不能正常执行
    - 回滚两个事务，数据项的锁都被释放
    - 大多数所协议都存在死锁的可能
  - 两阶段封锁协议
    - 保证可串行性的协议
    - 每个事物分两个阶段提出加锁和解锁申请
      - 增长阶段：事务可以获得锁，但不能释放锁
      - 缩减阶段：事务可以释放锁，但不能获得锁
    - 最初，事务处于增长阶段，一旦释放了锁，就处于缩减阶段
    - 并不能确保不会发生死锁
    - 有可能发生级联回滚
    - 两阶段是串行化的充分而非必要条件
  - 严格两阶段封锁协议——排他锁必须在事务提交之后才可以释放
    - 确保可恢复性并且避免级联回滚
  - 强两阶段封锁协议——事务提交之前不得释放任何锁
    - 按提交顺序串行化
- 死锁处理
  - 死锁预防、死锁检测、死锁恢复
  - 死锁预防
    - 对加锁请求进行排序或要求同时获得所有的锁来保证不会发生循环等待
    - 抢占与事务回滚
      - 每个事务赋一个唯一的时间戳，系统用时间戳来决定事务等待还是回滚
      - 若一个事务回滚，则该事务重启时保持原有时间戳

        - 两种死锁防御机制
            - wait-die：非抢占技术。事务1申请的数据项被事务2持有，当且仅当事务1的时间戳小于事务2，允许事务1等待，否则回滚
            - wound-wait：抢占技术。事务1申请的数据项被事务2占有，仅当事务1的时间戳大于事务2时，允许事务1等待，否则事务2回滚
        - 不必要的回滚
    - 锁超时，申请锁的事务至多等待给定时间，否则超时回滚
- 死锁检测
    - 等待图，边$T_i->T_j$表示i在等待j释放数据项
    - 有环时存在死锁
- 从死锁中恢复
    - 回滚一个或多个事务
    - 选择牺牲者
    - 回滚
        - 彻底回滚：中止事务，重新开始
        - 部分回滚：回滚到可以解除死锁
    - 饥饿
        - 我们要保证一个事务被选作牺牲者的次数有限
- 多粒度
    - 允许各种大小的数据项并定义数据力度的层次结构，小粒度数据项嵌套在大粒度数据项中实现

以上内容整理于 幕布文档