13 恢复系统

- 考点
  - Logging functionality 日志的概念，日志的作用
  - Basic recovery algorithm 基本的故障恢复算法
- 一班讲的
  - failure classification
  - storage structure
  - recovery and atomicity
  - log-based recovery
- 恢复算法
  - 在正常事务处理期间采取的措施，以确保存在足够的信息以从故障中恢复
  - 将数据库内容恢复到确保原子性、一致性和持久性的状态失败后采取的操作
- 故障分类
  - 事务故障
    - 逻辑错误：内部条件，非法输入、找不到数据、溢出等
    - 系统错误：系统死锁，但该事务可以在之后重新执行
  - 系统崩溃：硬件故障
  - 磁盘故障：磁头损坏等
- 存储结构
  - 易失性存储器
  - 非易失性存储器
  - 稳定存储器
- 恢复与原子性
  - 日志记录：日志是日志记录的序列，记录所有数据库中的更新活动
    - 记录数据库修改的结构——日志
    - 事务标识，执行write的唯一标识
    - 数据项标识，数据项在磁盘上的位置，快标识和块内偏移量
    - 旧值
    - 新值
    - $< T_i, X_j, V_1, V_2 >$事务i对X_j执行了一个写操作
    - start---开始，commit-----提交， abort------中止
  - 数据库修改
    - undo使用一个日志记录，将该日志记录中指明的数据项设置为旧值
    - redo使用一个日志记录，将该日志记录中指明的数据项设置为新值

- 使用日志进行redo（重做）和undo（撤销）操作
  - redo：对日志进行一次扫描，遇到一个redo日志记录就执行一次redo动作。确保保持更新的顺序，并且效率更高
  - undo：不仅将数据项恢复成它的旧值，而且是撤销过程的一个部分，写日记记录来记下所执行的更新。undo操作完之后，写一个$< T_i abort >$日志记录，表明撤销完成。对每个事务，undo只执行一次，正常处理中该事务回滚
  - 如果日志包括$< T_i start >$记录既不包括commit也不包括abort，需要对事务进行撤销
  - 包括start，commit或abort，需要对事物进行重做
- 检查点
  - 在执行检查点操作的过程中不允许执行任何更新
  - 在执行检查点的过程中将所有更新过的缓冲块都输出到磁盘
  - 执行过程
    - 将当前位于主存的所有日志记录输出到稳定存储器
    - 将所有修改的缓冲块输出到磁盘
    - 将一个日志记录<checkpoint L>输出到稳定存储器，其中L是执行检查点时正活跃的事务列表
  - 在<checkpoint L>之前的<commit>可以不做redo
  - 崩溃之后，系统检查日志找到最后一条<checkpoint L>记录（尾端反向搜索），只需要对L中的事务以及<checkpoint>写到日志之后才开始执行的事务进行undo或redo操作。
- 恢复算法
  - 事务回滚
    - 正常的事务回滚
      - 从后向前扫描日志，对于发现的T_i的每项日志记录，旧值写回，向日志中写一个特殊的只读日志，不需要undo信息、
      - 一旦发现start，就停止从后往前的扫描，并在日志中写一个abort
  - 崩溃后的恢复：重启时两阶段进行
    - 在redo阶段：从最后一个检查点开始正向扫描日志
      - 将 要回滚的事务列表undo-list初始设定为<checkpoint L>中的L
      - 一旦遇到正常日志形式或者redo-only记录，就重做这个操作
      - 一旦发现start，就把对应事务加到undo-list
      - 一旦发现abort或commit，就把对应事务从undo-list中去掉
    - 在undo阶段：系统回滚undo-list中的所有事务。通过尾端开始反向扫描日志来执行回滚
      - 一旦发现属于undo-list中的事务，就执行undo操作
      - 发现undo-list中事务的start，就像日志中写一个abort，并且从undo-list去除掉

- undo-list变为空表，找完了所有的start，撤销阶段结束

以上内容整理于 幕布文档