Machine Translated by Google

第十六章：16：恢复Recovery

数据库系统概念，第 7版。

©Silberschatz、Korth 和 Sudarshan
见www.db-book.com再利用条件

Machine Translated by Google

恢复算法 Recovery Algorithms

- 假设交易Ti将 50 美元从账户A转移到账户B
  - 两次更新：A 减 50，B 加 50
- 事务Ti需要对 A 和 B 的更新输出到数据库
  - 在进行其中一项修改之后但在进行这两项修改之前，可能会发生故障
  - 在未确保事务提交的情况下修改数据库可能会使数据库处于不一致状态
  - 如果在事务提交后发生故障，不修改数据库可能会导致丢失更新
- 恢复算法有两个部分
  1. 在正常交易处理过程中采取的措施，以确保足够存在从故障中恢复的信息
  2. 将数据库内容恢复到状态失败后采取的措施

     确保原子性、一致性和持久性

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

故障分类 Classification

- 交易失败：
  - 逻辑错误：由于某些内部错误条件，事务无法完成
  - 系统错误：数据库系统必须因错误情况（例如死锁）而终止活动事务
- 系统崩溃：电源故障或其他硬件或软件故障导致系统崩溃
  - 故障停止假设：系统将在出现故障时停止
    失败，并且假定非易失性存储内容不会因系统崩溃而损坏
- 磁盘故障：磁头崩溃或类似的磁盘故障会破坏全部或部分磁盘
  贮存
  - 假设破坏是可检测的：磁盘驱动器使用校验和来检测
    检测故障

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

存储结构 Structure

- 易失性存储： 无法在系统崩溃后幸存 示例：主内存、高速缓存
- 非易失性存储： 经受住系统崩溃 示例：磁盘、磁带、闪存、非易失性RAM 但仍可能发生故障，丢失数据
- 稳定存储： 一种神话般的存储形式，可在所有故障中幸存 近似于维护多个副本关于不同的非易失性媒体

Machine Translated by Google

数据访问 Data Access

- 物理块是驻留在磁盘上的那些块
- 缓冲块是临时驻留在主存中的块
- 磁盘和主存之间的块移动通过以下两个操作启动：
  - 输入(B)将物理块B传输到主存储器
  - 输出(B)将缓冲块B传输到磁盘，并替换那里的适当物理块
- 我们假设每个数据项都适合并存储在单个块中

Machine Translated by Google

数据访问

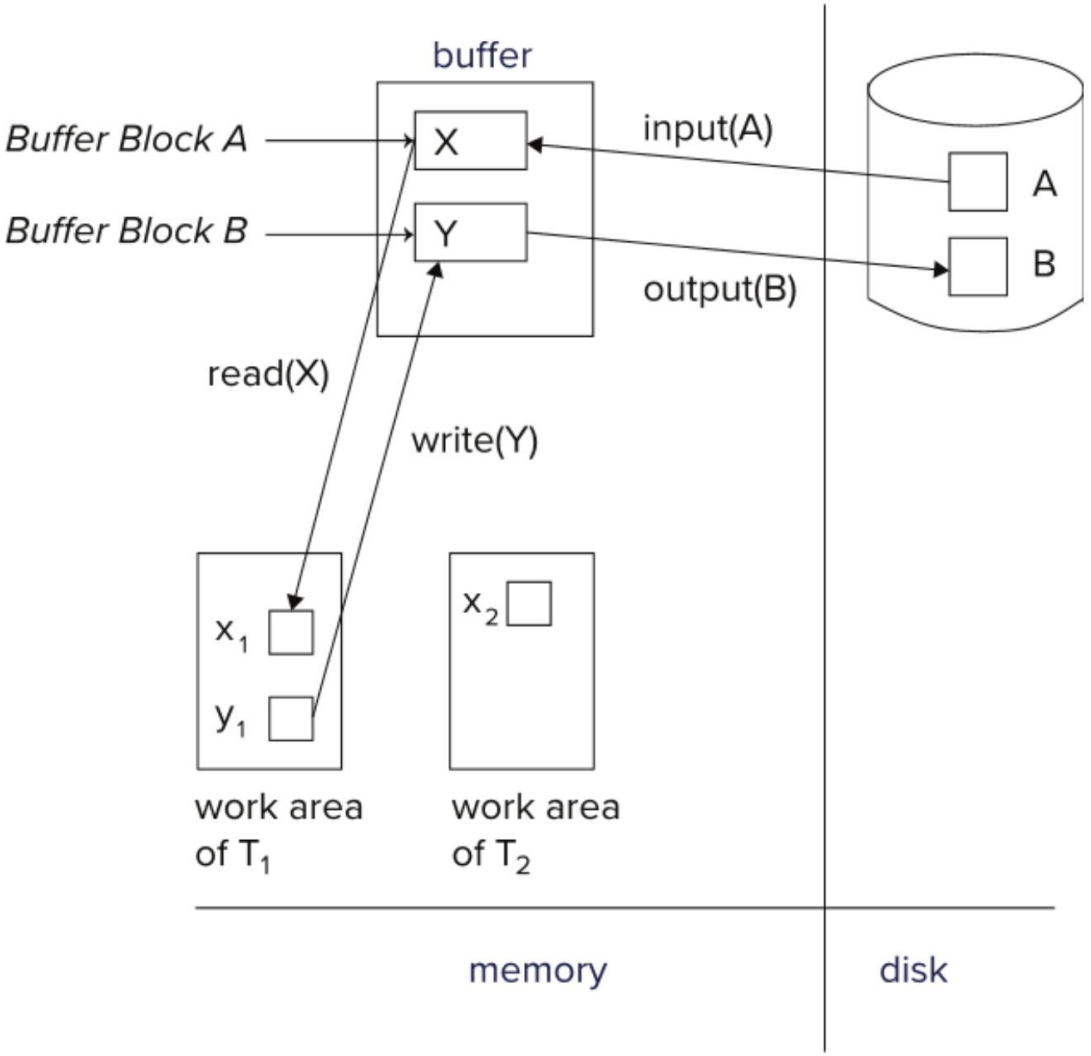
- 每个事务Ti在主存中都有它的私有工作区,由它访问和更新的所有数据项的本地副本被保存
  - Ti的数据项X的本地副本称为xi
- 在系统缓冲块及其私有工作之间传输数据项

  完成区域:
  - read(X)将数据项X的值赋给局部变量xi
  - write(X)将局部变量xi的值赋给
    缓冲块
  - 注意:输出（BX)不需要紧跟在写（X)之后
    - 系统可以在它认为合适的时候执行输出操作
- 交易
  - 必须在第一次访问X之前执行read(X) （后续读取可以来自本地副本）
  - write(X)可以随时执行

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

数据访问示例 Data Access



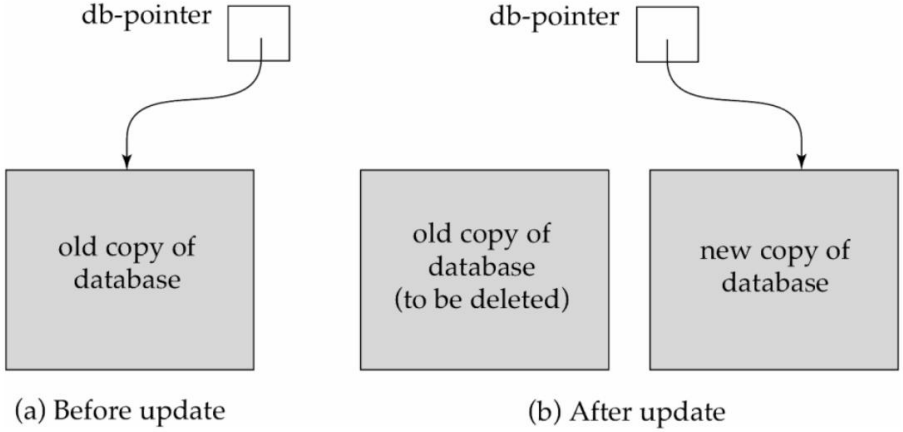
©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

恢复和原子性 Recovery and Atomicity

- 为确保故障的原子性，我们首先输出描述对稳定存储的修改的信息，而不修改数据库本身
- 我们将关注基于日志的恢复机制
- 较少使用的替代方案：影子复制和影子分页

影子副本（原始副本）



©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

基于日志的恢复 Log-Based Recovery

- 日志是一系列日志记录。记录在数据库中保存有关更新活动的信息

  - 日志保存在稳定的存储中
- 当事务Ti开始时，它通过写一个

  <Ti start>日志记录

- 在Ti执行write(X) 之前，一条日志记录

  <钛、X、 V1 、 V2>

  被写入，其中V1是X写入前的值（旧值或前映像）， V2是要写入X的值（新值或后映像）

- 当Ti完成最后一条语句时，日志记录<Ti commit>被写入

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

数据库修改 Database Modification

- 如果事务在磁盘缓冲区或磁盘本身上执行更新，我们说事务会修改数据库
  - 对主存私有部分的更新不算作数据库修改
- 立即修改方案允许更新
  在事务提交之前对缓冲区或磁盘本身进行未提交的事务
- 必须在写入数据库项之前写入更新日志记录
  - 日志记录直接输出到稳定存储
- 更新块输出到磁盘可以在任何时间之前或
  事务提交后
- 延迟修改方案仅在事务提交时更新缓冲区/磁盘

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

提交点

- 当所有事务操作对数据库的影响都已输出到日志时,就说事务到达了它的提交点
- 当一个事务的提交日志记录被输出到稳定存储时,就说它已经提交（超出提交点）
  - 事务的所有先前日志记录也必须已输出
    已经
- 事务提交时,事务执行的写入可能仍在缓冲区中,稍后可能会输出

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

数据库修改示例

日志	写	输出
<T0开始>		
<T0，一、1000、950>		
<T0，乙、2000、2050>		
	A = 950 B = 2050	
<T0提交>		
<T1开始>		
<T1、C、700、600>		
	C = 600	
		BB _ _
<T1提交>		
		不是

T1提交前的BC输出

T0提交后的BA输出

- 注意： BX表示包含X的块。

Machine Translated by Google

撤消和重做操作

- 撤消和重做事务
  - undo(Ti) -- 将Ti更新的所有数据项的值恢复到它们的旧值,从Ti的最后一条日志记录返回
    - 每次数据项X恢复到其旧值V时,都会写出一个特殊的日志记录<Ti , X, V>
    - 当一个事务撤消完成时,一个日志记录
      <Ti abort>被写出
  - redo(Ti) -- 将Ti更新的所有数据项的值设置为新值,从Ti的第一条日志记录开始

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

从失败中恢复 Recovering from Failure

- 故障恢复时：
  - 如果日志包含记录<Ti start>，则需要撤消事务 Ti，
    - 但不包含记录<Ti commit>或<Ti abort>。
  - 事务Ti需要为所有已完成的事务重做，如果日志包含记录<Ti start>
    - 并包含记录<Ti commit>或 <Ti abort>

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

从失败中恢复 Recovering from Failure

- 假设失败的事务Ti在系统故障之前被撤消,并且 <Ti abort>记录被写入日志,然后出现系统故障
- 从系统故障中恢复时,事务Ti被重做
  - 这种重做重做事务Ti的所有原始操作,包括恢复旧值的（回滚)步骤
    - 被称为重复历史
    - 进行所有更改然后回滚似乎很浪费,重复失败事务的历史记录简化了恢复

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

恢复示例 Recovery Example

在这里,我们显示了在失败前的三个时间实例中出现的日志。

(a)	(b)	(c)
$<T_0$ start>	$<T_0$ start>	$<T_0$ start>
$<T_0, A, 1000, 950>$	$<T_0, A, 1000, 950>$	$<T_0, A, 1000, 950>$
$<T_0, B, 2000, 2050>$	$<T_0, B, 2000, 2050>$	$<T_0, B, 2000, 2050>$
	$<T_0$ commit>	$<T_0$ commit>
	$<T_1$ start>	$<T_1$ start>
	$<T_1, C, 700, 600>$	$<T_1, C, 700, 600>$
		$<T_1$ commit>

上述每种情况下的恢复动作为： (a) undo

(T0)： B恢复为 2000， A恢复为 1000,并记录<T0 , B, 2000>, <T0 , A, 1000>, <T0 , abort>被写出 (b) 重做(T0)：A和B分别设置为 950 和 2050 和撤消(T1)： C恢复设置为 日志记录<T1 , C, 700>, <T1 , abort>被写出(c) 重做(T0) 和重做(T1)： A和B分别设置为 950 和 2050

分别， C设置为 600

Machine Translated by Google

检查站 Checkpoints

- 重做/撤消日志中记录的所有事务可能非常缓慢
  - 如果系统运行了一段时间,则处理整个日志非常耗时很久
  - 我们可能会不必要地重做已经将更新输出到数据库的事务

- 通过定期执行检查点来简化恢复程序，在。。。期间
  1. 检查点操作正在进行时所有更新都停止
  2. 将当前驻留在主存中的所有日志记录输出到stable贮存
  3. 将所有修改后的缓冲块输出到磁盘
  4. 输出一条日志记录< checkpoint L>到stable storage,其中L是一个检查点时所有活跃事务的列表

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

检查站 Checkpoints

- 考虑在检查点之前完成的事务Ti
- 对于这样的事务， <Ti commit>记录（或<Ti abort>）
  记录出现在< checkpoint L>记录之前的日志中
- Ti所做的任何数据库修改必须在检查点之前或作为检查点本身的一部分写入数据库

- 因此,在恢复时,无需执行重做操作
  关于钛

©Silberschatz、Korth 和 Sudarshan

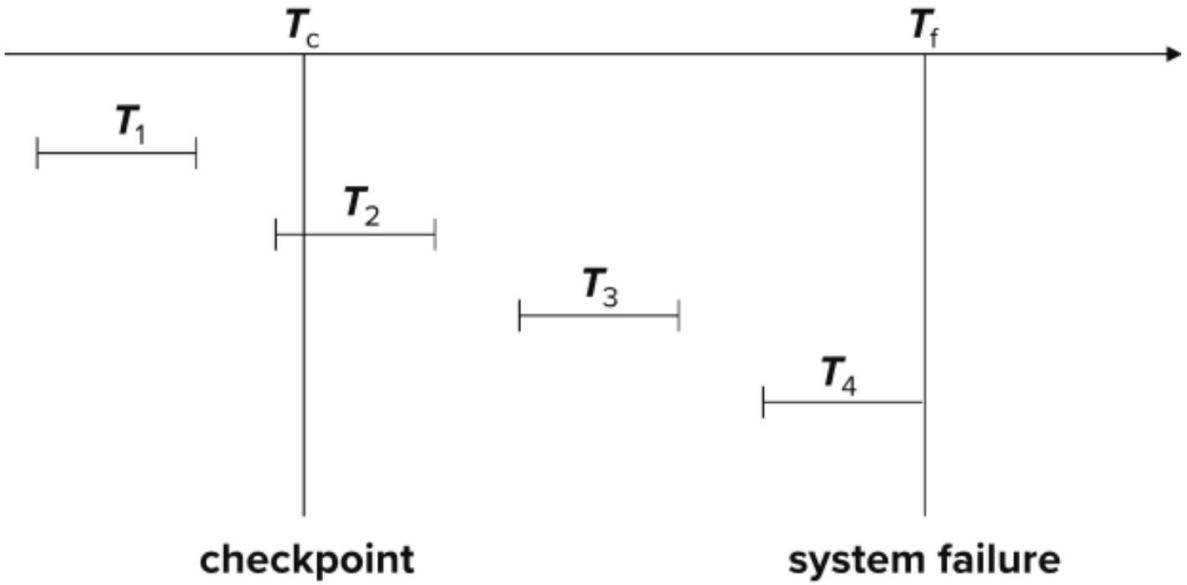
Machine Translated by Google

检查站 Checkpoints

- 发生系统崩溃后,系统检查日志以查找最后一个<checkpoint L>记录（这可以通过向后搜索日志来完成,从日志末尾开始,直到第一个<checkpoint L>记录找到）
- 重做或撤消操作只需要应用于 L 中的事务,以及在<checkpoint之后开始执行的所有事务 L>记录被写入日志。让T表示这组交易。
  - 对于T中没有<Tk commit>记录或<Tk的所有事务Tk abort>记录在日志中,执行 undo(Tk)
  - 对于T中的所有事务Tk ,使得记录<Tk commit>或记录<Tk abort>出现在日志中,执行 redo(Tk)
- 也就是说,我们只需要检查从最后一个检查点日志记录开始的日志部分,就可以找到事务集T并找出 T 中每个事务的日志中是否出现了提交或中止记录

Machine Translated by Google

检查点示例



- T1可以忽略（由于检查点,更新已经输出到磁盘）
- T2和T3重做
- T4撤消

Machine Translated by Google

恢复算法

- 记录（在正常操作期间）：
  - <Ti start>
  - 在事务开始时
  - <Ti , Xj , V1 , V2>用于每次更新，以及
  - <Ti commit>在事务结束时
- 事务回滚（在正常操作期间）

- 令Ti为要回滚的事务
- 从末尾向后扫描日志，对于Ti的每条日志记录，格式为<Ti , Xj , V1 , V2>
- 通过将V1写入Xj来执行撤消，
- 写一个日志记录<Ti , Xj , V1>
- 这样的日志记录称为补偿日志记录

- 一旦找到记录<Ti start> ，停止扫描并写入日志记录<Ti abort>

- 观察由事务或代表事务执行的每个更新操作，包括为将数据项恢复到其旧值而采取的操作，现已记录在日志中

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

恢复算法 Recovery Algorithm

- 从故障中恢复:两个阶段
  - 重做阶段:重放所有事务的更新,无论它们是否已提交、中止或不完整
  - 撤消阶段:撤消所有未完成的事务
- 重做阶段:
  1. 找到最后一条<checkpoint L>记录,设置undo-list为L
  2. 从< checkpoint L >记录1上方向前扫描_ _

     遇到<Ti , Xj , V1>形式的补偿日志记录,操作重做;即,将值V1写入数据项 Xj

     2.每当找到一条日志记录<Ti start> ,将Ti添加到undo-list 3.每当找到一条日志记录<Ti commit>或<Ti abort> ,
     从 undo-list中删除Ti （不需要回滚已中止的Ti ,因为在其中有<Ti abort> ,这表明它已经成功中止）

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

恢复算法 Recovery Algorithm

- 撤消阶段：

从末尾向后扫描日志

1.每当在 undo-list中找到Ti的日志记录<Ti , Xj , V1 , V2>时,执行与事务回滚相同的操作： 1.通过将V1写入Xj来执行撤消。 2.写日志记录 <Ti , Xj , V1>

2.每当在 undo-list 中找到Ti所在的日志记录<Ti start>时，

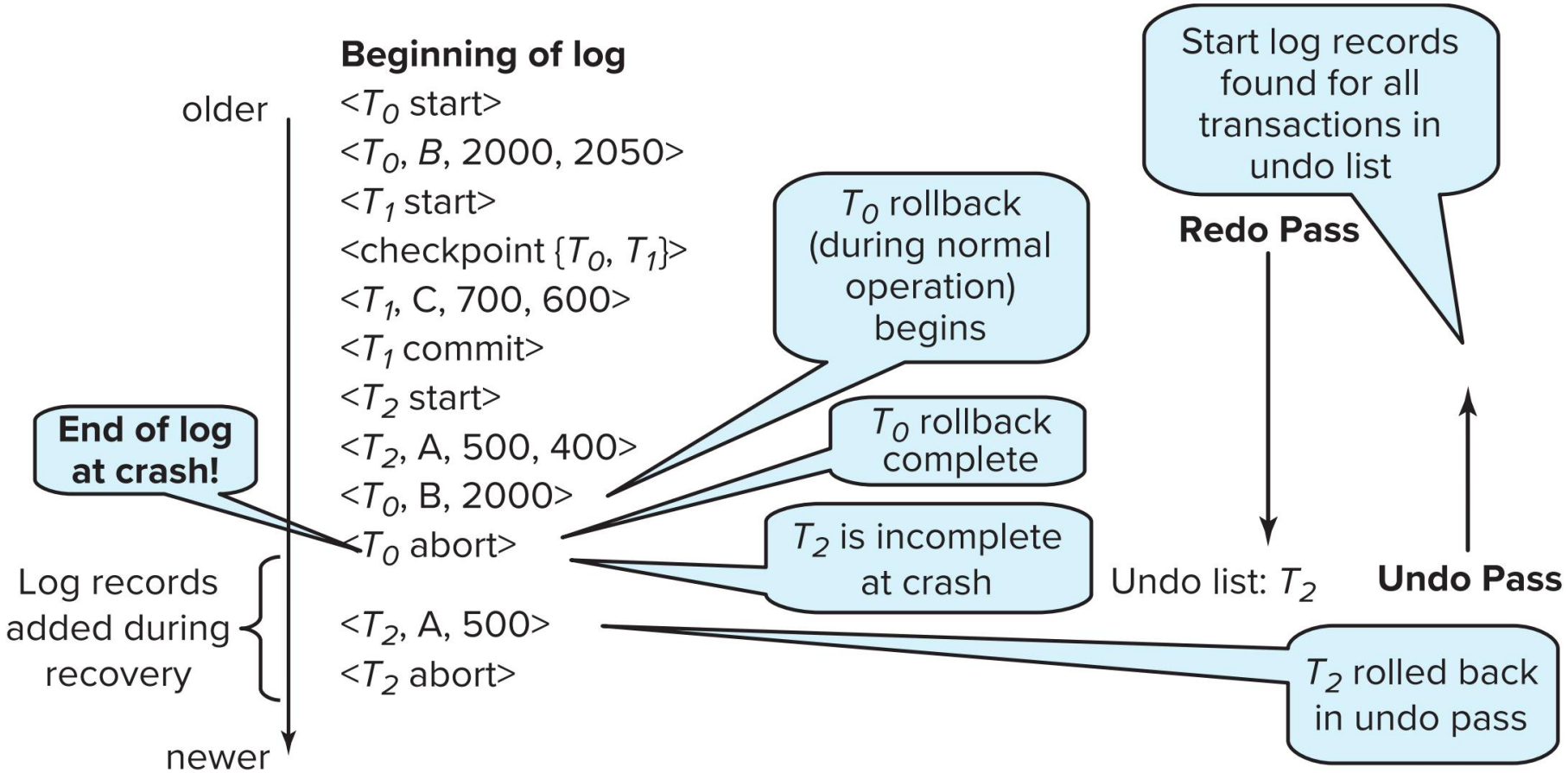
1.写一条日志记录<Ti abort> 2.从 undo-list 中移除Ti 3.当 undo-list 为空时停止

1.即undo-list 中的每一个事务都找到了 <Ti start> （因为如果没有找到 < Ti start> ,那么Ti就不能从 undo-list 中删除,undo-list 也不会被删除）空） - 撤消阶段完成后,可以开始正常的事务处理

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

恢复示例



Machine Translated by Google

日志记录缓冲 Log Record Buffering

- 日志记录缓冲：日志记录缓冲在主内存中，而不是直接输出到稳定存储
  - 当缓冲区中的一个日志记录块已满，或者执行了日志强制操作时，日志记录被输出到稳定存储
- 日志强制通过强制其所有日志记录（包括提交记录）到稳定存储来提交事务

©Silberschatz、Korth 和 Sudarshan

Machine Translated by Google

日志记录缓冲 Log Record Buffering

- 如果日志记录被缓冲,则必须遵循以下规则:
  - 日志记录按顺序输出到稳定存储
    创建
  - 事务Ti仅在日志记录时才进入提交状态
    <Ti commit>已输出到稳定存储
  - 在主存中的数据块输出到数据库之前,与该块中的数据相关的所有日志记录必须已经输出到稳定存储
    - 此规则称为预写日志记录或WAL规则