实现篇
第十一章 数据库恢复

主讲：程思瑶
海量数据计算研究中心

研究动机

- 数据库恢复机制用来进行事务终止后的善后处理和系统故障恢复处理，确保系统发生故障以后，数据库能够恢复到正确的状态。

目录

- 数据库恢复的必要性
- 使用日志的数据库恢复技术
- 使用检查点的数据库恢复技术
- 恢复算法
- 缓冲技术

目录

- 数据库恢复的必要性
- 使用日志的数据库恢复技术
- 使用检查点的数据库恢复技术
- 恢复算法
- 缓冲技术

数据库恢复的必要性

- 数据库系统的数据库恢复机制的目的有两个：
  - 第一个目的是保证事务的原子性，即确保一个事务被交付运行之后，要么该事务中的所有数据库操作都被成功地完成而且这些操作的结果被永久地存储到数据库中，要么这个事务对数据库没有任何影响。
  - 第二个目的是当系统发生故障以后，数据库能够恢复到正确状态。

数据库恢复的必要性

- 故障分类
  - 事务故障
    - 逻辑故障：事务由于某些内部条件而无法继续正常执行，如非法输入、找不到数据、溢出或超出资源限制
    - 系统错误：系统进入一种不良状态，如死锁，结果事务无法继续正常执行，但该事务可以在以后的某个时间重新执行
  - 系统崩溃
    - 硬件故障，或者数据库软件或操作系统的漏洞，导致易失性存储器内容的丢失，并使得事务处理停止，而非易失性存储器仍完好无损
  - 磁盘故障
    - 在数据传送操作过程中由于磁头损坏或故障造成磁盘块上的内容丢失

数据库恢复的必要性

- 故障恢复与处理
  - 首先，确定用于存储数据的设备的故障方式
  - 其次，必须考虑这些故障方式对数据库的影响
  - 然后，提出在故障发生后仍保持数据库一致性及事务原子性的算法，主要分两部分
    - 在正常事务处理时采取措施，保证有足够的信息可用于故障恢复
    - 故障发生后采取措施，将数据库内容恢复到某个保证数据库一致性，事务原子性及持久行的状态。

目录

- 数据库恢复的必要性
- **使用日志的数据库恢复技术**
- 使用检查点的数据库恢复技术
- 恢复算法
- 缓冲技术

使用日志的数据库恢复技术

- 数据库系统日志
  - 记录有关事务的数据库操作信息的存储结构是数据库系统日志，简称日志。
  - 日志记录的格式：
    - <T, start >，表示事务T已经开始
    - < T, commit>，表示事务T成功完成
    - <T, abort>，事务T未成功，被中止
    - <T, X, v1,v2>，表示事务T改变了数据库元素X，X原来的值为v1(X的旧值),v2为新值
  - 为了保证日志在系统和磁盘发生故障时仍可使用，它必须永久地存在磁盘上。

使用日志的数据库恢复技术

- 无论什么时候，当一个事务执行完一个写操作WRITE(Q)，就应该在数据库被修改之前建立起描述这个写操作的日志记录。

- 需要时，既可以用日志记录中存储的Q的新值更新数据库中的Q，也可以根据日志记录中把数据库中已经由WRITE(Q)操作改变的Q值恢复到WRITE(Q)执行之前的值。

使用日志的数据库恢复技术

- 使用日志的数据库恢复技术
  - 推迟更新技术
  - 即时更新技术

使用日志的数据库恢复技术

- 推迟更新技术
  - 为了保证事务的原子性，在每个事务运行期间，推迟更新技术在日志中记录这个事务对数据库的所有更新操作，**把所有数据库更新操作推迟到该事务提交时执行**。
  - 推迟更新技术必须遵循下述**推迟更新协议**：
    - (1) 每个事务在到达提交点之前不能更新数据库。
    - (2) 在一个事务的所有更新操所作对应的日志记录永久写入存储器之前，该事务不能到达提交点。

使用日志的数据库恢复技术

- 推迟更新技术
  - 当一个事务到达提交点时，称该事务进入部分提交状态。
  - 推迟更新协议保证当一个事务部分提交时，这个事务的所有更新操作的信息都已记录在日志中。
  - 当一个事务部分提交时，推迟更新技术可以使用日志中有关该事务的数据库更新操作的信息更新数据库。如果在一个事务部分提交之前异常结束或系统发生故障，日志中有关这个事务的信息将被删除。

使用日志的数据库恢复技术

- 考虑银行数据库系统：
  - 事务T_0从帐号A向帐号B转储50元钱：

    T_0: read(A);
    A := A - 50;
    write(A);
    read(B);
    B := B + 50;
    write(B)。
  - 事务T_1从帐号C支出100元钱：

    T_1: read(C);
    C := C - 100;
    write(C)。

使用日志的数据库恢复技术

- 考虑银行数据库系统：
  - 设A、B和C的初值分别是1000、2000、和700元，并且T_0和T_1按串行调度$<T_0, T_1>$执行。日志中包含的有关T_0和T_1的信息如下：

$<T_0$, start $>$	$<T_1$, star $>$
$<T_0$, A, 950$>$	$<T_1$, C, 600$>$
$<T_0$, B, 2050$>$	$<T_1$, commmit $>$
$<T_0$, commit $>$	

使用日志的数据库恢复技术

- 考虑银行数据库系统：
  - 按照推迟更新协议，T_0和T_1的执行结果写入数据库和日志的一种顺序：

日志文件	数据库
$<T_0$, start >	
$<T_0$, A, 950>	
$<T_0$, B, 2050>	
$<T_0$, commit >	A = 950 B = 2050
$<T_1$, star >	
$<T_1$, C, 600>	
$<T_1$, commmit >	C = 600

使用日志的数据库恢复技术

- 使用日志，数据库恢复机制可以处理任何导致非永久存储器的信息丢失问题。
  - 数据库恢复机制需要下边的操作：

    redo(T): FOR 日志中每个型为<T, X, V>的记录 DO

    把数据库中数据项X的值改为值V;

    ENDFOR。
  - redo操作必须是幂等的，即执行多次和执行一次的效果相同。

使用日志的数据库恢复技术

- 当系统发生故障并被修复以后，数据库恢复机制将考察日志，确定需要重做的事务T。
  - 事务T需要重做当且仅当日志包含记录<T, start>和<T, commit>。
  - 如果系统在事务T成功完成之后发生故障，日志中有关T的信息将被用来将数据库恢复到正确状态。

使用日志的数据库恢复技术

- 使用推迟更新技术的数据库恢复过程定义如下：
  - (1) 从后向前扫描日志记录，建立两个事务表：一个表称为提交事务表，包含全部具有日志记录<T, commit>的事务T，即已提交的事务；另一个表称为未提交事务表，包括全部具有日志记录<T, start>，但不具有日志记录<T, commit>的事务T，即尚未提交的事务；
  - (2) 对于提交事务表中的每个事务T，执行REDO(T)；
  - (3) 对于未提交事务表中的每个事务T，删除所有T的日志记录，放弃T，待以后重新启动执行。

使用日志的数据库恢复技术

- 事务T0和T1定义如前，而且按调度<T0, T1>运行。

日志文件	数据库
$<T_0,$ start $>$	
$<T_0,$ A, 950$>$	
$<T_0,$ B, 2050$>$	
$<T_0,$ commit $>$	
	A = 950 B = 2050
$<T_1,$ star $>$	
$<T_1,$ C, 600$>$	
$<T_1,$ commmit $>$	
	C = 600

使用日志的数据库恢复技术

- 假定系统在事务运行结束之前出现故障。
  - 设故障恰好发生在T0的write(B)操作的信息被写入日志之后。

<T0, start>
<T0, A, 950>
<T0, B, 2050>

  - 在这种情况下，数据库恢复机构不必采取任何恢复行动，因为日志中没有提交记录。A和B的值仍然保持为1000和2000元。

使用日志的数据库恢复技术

- 假定系统在事务运行结束之前出现故障。
  - 设故障恰好发生在write(C)执行之后。

<T0, start>
<T0, A, 950>
<T0, B, 2050>
<T0, commit>
<T1, start>
<T1, C, 600>

  - 这种情况下，数据库恢复机构需要执行redo(T0)操作，因为记录<T0, commit>在日志中。当redo(T0)操作被执行之后，帐号A和B的值分别是950和2050元。帐号C的值仍然是700元。

使用日志的数据库恢复技术

- 假定系统在事务运行结束之前出现故障。
  - 设故障恰好发生在<T1, commits>执行之后。

<T0, start>
<T0, A, 950>
<T0, B, 2050>
<T0, commit>
<T1, start>
<T1, C, 600>
<T1, commit>

  - 这种情况下，由于日志中包含两个提交记录<T0, commit>和<T1, commit>，数据库恢复机制必须执行redo(T0)和redo(T1)。
  - 这些操作执行完以后，帐号A、B和C的值分别是950、2050和600。

使用日志的数据库恢复技术

- 再考虑另一种情况：
  - 正在恢复时又发生了第二次系统故障。
  - 由于已经执行了一些redo操作，数据库发生了部分更新，但有可能并没将所有的数据库更新都记入数据库。
  - 当系统的第二次故障到来时，恢复工作与上面的例子相同，对于日志中的每个提交记录<T_i, commit>，执行redo(T_i)。

使用日志的数据库恢复技术

- 使用日志的数据库恢复技术
  - 推迟更新技术
  - 即时更新技术

使用日志的数据库恢复技术

- 即时更新技术
  - 即时更新技术允许事务直接更新数据库。处于活动状态的事务直接在数据库上实施的更新称为非提交更新。任何即时更新技术都必须遵循如下的即时更新协议：
    - (1) 所有<T, X, v_1,v_2>型日志记录安全地、永久存储到存储器之前，事务T不能更新数据库。
    - (2) 所有<T, X, v_1,v_2>型日志记录安全地、永久存储到存储器之前，不允许事务T提交。

使用日志的数据库恢复技术

- 即时更新协议保证在系统故障发生时，每个运行事务的更新操作的描述信息**都安全地记录在日志中**。
- 一旦系统故障导致事务T失败，即时更新技术将根据<T, X, v_1, v_2>型日志记录，把数据项X的值恢复为它的原始值v_1。

使用日志的数据库恢复技术

- 即时更新协议
  - T开始执行时，记录<T, start>被写到日志。
  - 在T运行期间，当T发出一个write(X)操作时，记录<T, X, v_1, v_2>首先被写入日志，然后直接在数据库上执行write(X)。
  - 当T部分提交时，记录<T, commit>被写到日志。

使用日志的数据库恢复技术

- 仍以事务T0和T1为例说明即时更新技术。
  - 设事务T0和T1按调度<T0, T1>顺序执行。日志中有关这两个事务的记录如下：

<T0, start >
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit >
<T1, start >
<T1, C, 700, 600>
<T1, commit >

使用日志的数据库恢复技术

- 当T0和T1运行时，数据库和日志按照即时更新协议变化过程：

日志文件	数据库
<T0, start >	
<T0, A, 1000, 950>	
	A=950
<T0, B, 2000, 2050>	
	B=2050
<T0, commit >	
<T1, start >	
<T1, C, 700, 600>	
	C=600
<T1, commit >	

使用日志的数据库恢复技术

- 即时更新技术需要如下两个操作：
  - (1) undo(T):

    FOR 日志中每个型为<T, X, V_1, V_2>的记录 DO

    把数据库中数据项X的值改为V_1;

    ENDFOR。
  - (2) redo(T):

    FOR 日志中每个型为<T, X, V_1, V_2>的记录 DO

    把数据库中数据项X的值改为V_2;

    ENDFOR。
- undo和redo操作必须是幂等的，即执行多次和执行一次的效果相同。

使用日志的数据库恢复技术

- 系统发生故障后，即时更新技术将调用如下的过程进行数据库的恢复处理：
  - (1) 从后向前扫描日志记录，建立两个事务表：一个表称为提交事务表，包含全部具有日志记录<T, commit>的事务T，即已提交的事务；另一个表称为未提交事务表，包括全部具有日志记录<T, start>，但不具有日志记录<T, commit>的事务T，即尚未提交的事务；
  - (2) 对于未提交事务表中的每个事务T，执行UNDO(T)，写一个<T,abort>日志记录，表明撤销完成；
  - (3) 对于提交事务表中的每个事务T，执行REDO(T)。

使用日志的数据库恢复技术

- 仍然假定事务T0和T1按照调度<T0, T1>顺序运行。日志中有关这两个事务的记录如下：

<T0, start >
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit >
<T1, start >
<T1, C, 700, 600>
<T1, commit >

使用日志的数据库恢复技术

- 假定系统在事务运行结束之前出现故障。
  - 设故障发生在T0的write(B)的日志记录已被写到永恒存储器后，在<T0, commit>记录写入日志之前。

<T0, start >
<T0, A, 1000, 950>
<T0, B, 2000, 2050>

  - 在这种情况下，数据库恢复机制必须执行undo(T0)操作。结果，数据库中帐号A和B的值被恢复成为1000和2000元。

使用日志的数据库恢复技术

- 假定系统在事务运行结束之前出现故障。
  - 设故障恰好发生在T1的write(C)的日志记录刚刚被写到存储器。

<T0, start >
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit >
<T1, start >
<T1, C, 700, 600>

  - 在这种情况下，由于日志中有<T1, start>记录，但没有<T1, commit>，数据库恢复机制必须执行undo(T1)；
  - 又由于日志中包含<T0, start>和<T0, commit>记录，恢复机制必须执行redo(T0)。
  - 恢复过程结束后，数据库中数据项A、B和C的值分别是950、2050和700元。
  - 注意，一般需要在redo操作之前执行undo操作。

使用日志的数据库恢复技术

- 假定系统在事务运行结束之前出现故障。
  - 设故障恰好发生在日志记录<T1, commits>被写到存储器之后。

```
<T0, start >
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
  <T0, commit >
  <T1, start >
  <T1, C, 700, 600>
    <T1, commit >
```

  - 在这种情况下，因为日志包含记录<T0, start>、<T0, commit>、<T1, start>和<T1, commit>，数据库恢复机制需要执行redo(T0)和redo(T1)。
  - 结果，数据库中的帐号A、B和C的值分别是950、2050和600元。

目录

- 数据库恢复的必要性
- 使用日志的数据库恢复技术
- **使用检查点的数据库恢复技术**
- 恢复算法
- 缓冲技术

使用检查点的数据库恢复技术

一、问题的提出

二、检查点技术

三、利用检查点的恢复策略

问题的提出

- 当系统故障发生时，我们必须检查日志，决定哪些事务需要重做，哪些需要撤销，存在困难是
  - 搜索过程太耗时
  - 大多数需要重做的事务已把其更新写入数据库中，尽管对它们重做不会造成不良后果，但会使恢复过程变得更长

检查点技术

- 具有检查点（checkpoint）的恢复技术
  - 在日志文件中增加检查点记录（checkpoint）
  - 恢复子系统在登录日志文件期间动态地维护日志
- 检查点的执行过程如下
  - 将当前位于主存的所有日志输出到稳定存储器
  - 将所有修改的缓冲块输出到磁盘
  - 将一个日志记录<checkpoint L>输出到稳定的存储器，其中L是执行检查点时正活跃的事务列表
- 在检查点执行过程中，不允许事务执行任何更新

检查点技术

- 恢复子系统可以定期或不定期地建立检查点,保存数据库状态
  - 定期
    - 按照预定的一个时间间隔，如每隔一小时建立一个检查点
  - 不定期
    - 按照某种规则，如日志文件已写满一半建立一个检查点

检查点技术

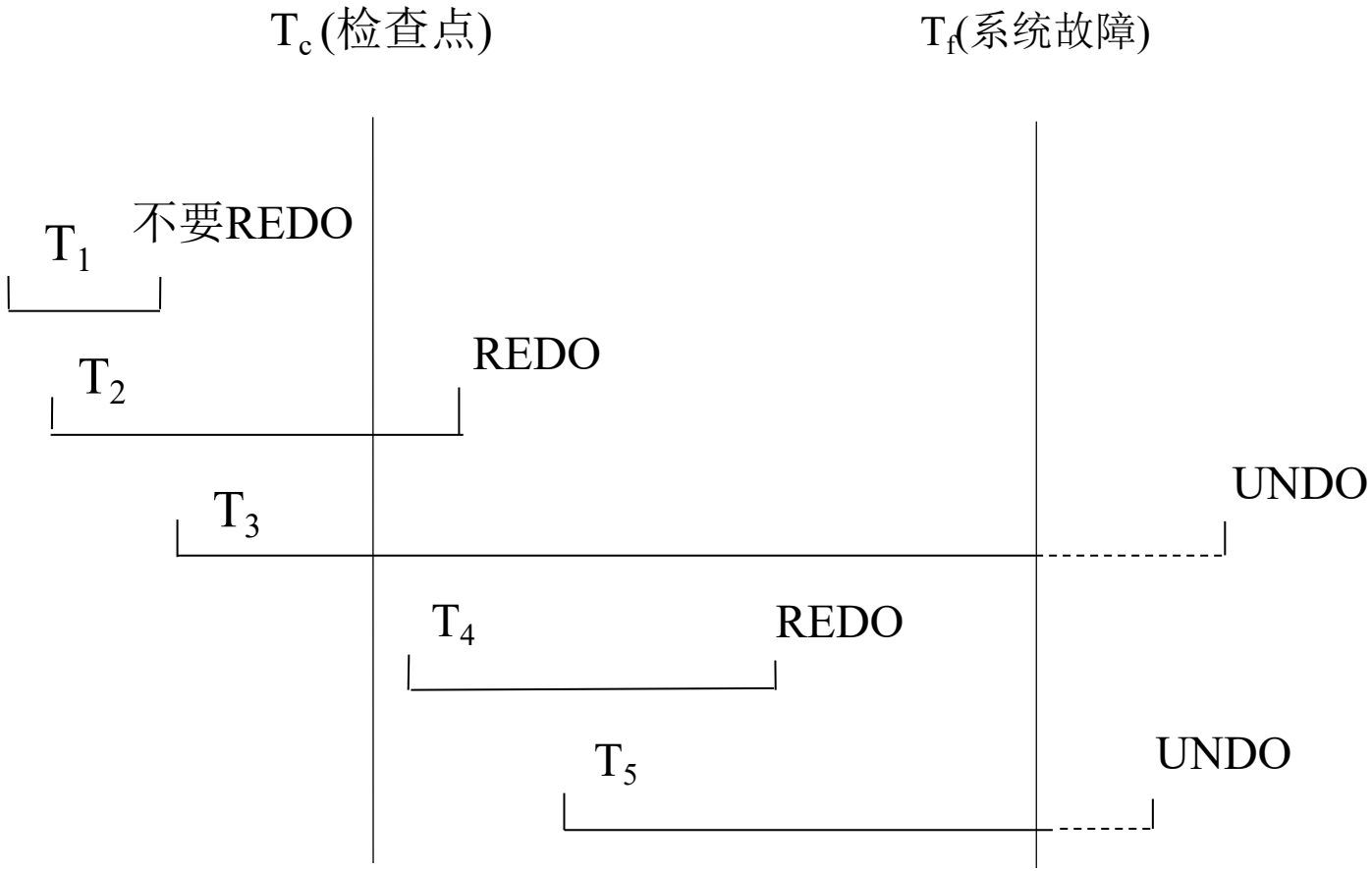
- 使用检查点技术的优势
  - 日志中加入<checkpoint L>记录使得系统恢复过程的效率得以提高
    - 在检查点前完成的事务T_i，所做的任何数据库修改都必然已在检查点前或作为检查点本身的一部分写入数据库中，因此不必再对T_i执行redo操作
    - 执行redo、undo只需考虑L list中涉及的事务及<checkpoint L>记录写到日志中之后才开始执行的事务

检查点技术

- 在系统崩溃发生之后，恢复方法如下
  - 系统检查日志以找到最后一条<checkpoint *L*>记录
  - 对L中的事务，以及<checkpoint *L*>记录写到日志中之后才开始执行的事务，做如下操作
    - 对于满足上述要求的T_k，若日志中没有<T_k, commit>或<T_k, abort>记录，则执行undo(T_k)
    - 否则执行redo(T_k)

检查点技术

系统出现故障时，恢复子系统将根据事务的不同状态采取不同的恢复策略



检查点技术

- T1：在检查点之前提交
- T2：在检查点之前开始执行，在检查点之后故障点之前提交
- T3：在检查点之前开始执行，在故障点时还未完成
- T4：在检查点之后开始执行，在故障点之前提交
- T5：在检查点之后开始执行，在故障点时还未完成

恢复策略：

- T3和T5在故障发生时还未完成，所以予以撤销
- T2和T4在检查点之后才提交，它们对数据库所做的修改在故障发生时可能还在缓冲区中，尚未写入数据库，所以要REDO
- T1在检查点之前已提交，所以不必执行REDO操作

目录

- 数据库恢复的必要性
- 使用日志的数据库恢复技术
- 使用检查点的数据库恢复技术
- **恢复算法**
- 缓冲技术

恢复算法

- 恢复算法的设计所需考虑一下两种情况
  - 正常操作时，事务如何回滚？
  - 系统崩溃时，如何进行事务的重做和撤销？

恢复算法

- 正常操作时，事务T_i的回滚执行如下操作
  - 从后往前扫描日志，对于所发现的T_i的每个形如$<T_i,X_j,V_1,V_2>$的日志记录
    - 值V_1被写到数据项X_j中，并且
    - 往日志中写一个特殊的只读日志记录$<T_i,X_i,V_1>$，称为补偿日志记录(compensation log record)，这样的记录不需要undo
  - 一旦发现$<T_i$, start$>$日志记录，就停止从后往前的扫描，并在日志中写一个$<T_i$, abort$>$

恢复算法

- 系统崩溃后的恢复
  - 重做阶段
  - 撤销阶段

恢复算法

- 重做阶段
  - 系统通过从最后一个检查点开始正向扫描日志
  - 将undo-list初始设为<checkpoint L>日志记录中L列表
  - 一旦遇到<T_i,X_j,V_1,V_2>的正常日志记录或形为<T_i,X_j,V_2>的redo-only日志记录，重做这个操作，将V_2的值写给数据项X_j
  - 一旦发现形为<T_i,start>的日志记录，将T_i加到undo-list
  - 一旦发现形为<T_i,commit>或<T_i,abort>的日志记录，将T_i从undo-list删除

恢复算法

- 撤销阶段(回滚undo-list的所有事务)
  - 从尾端开始反向扫描日志
  - 一旦发现属于undo-list中的事务的日志记录，就执行undo操作，如事务回滚算法
  - 当系统发现undo-list中事务Ti的<T_i,start>的日志记录，就往日志中写一个<T_i, abort>，并且将T_i从undo-list中去掉
  - 一旦undo-list为空表，即系统找到了开始时位于undo-list中的所有事务<T_i,start>，则撤销阶段结束

- Example

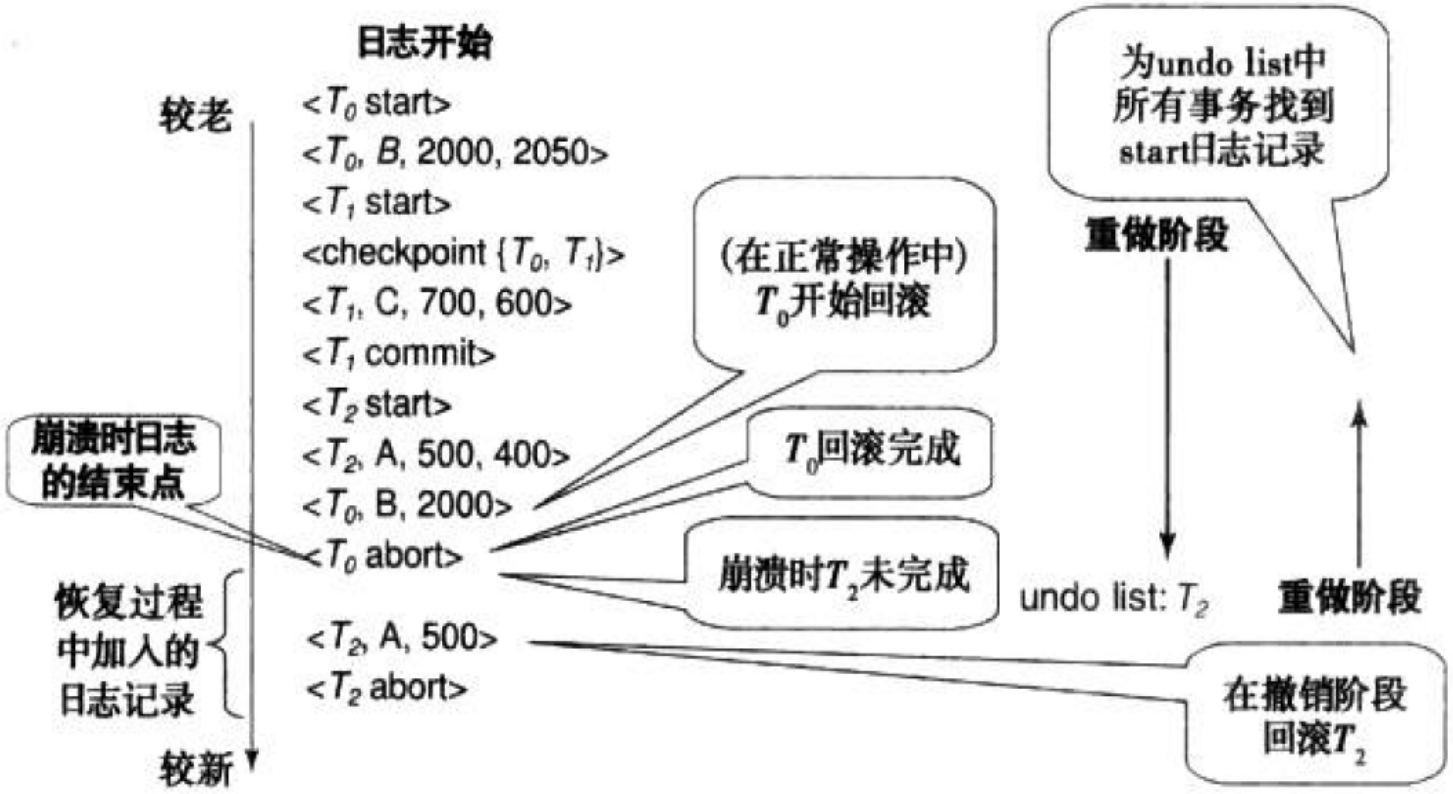


图 16-5　记录在日志中的动作和恢复中的动作的例子

恢复算法

设一个数据库系统启动后中，执行4个事务T0、T1、T2和T3。四个事务的内容如下：

T0: A := A + 20 (读入数据库元素A的值，加上20后，再写回A的值)

T1: B := B – 10 (读入数据库元素B的值，减去10后，再写回B的值)

T2: C := C * 2 (读入数据库元素C的值，乘以2后，再写回C的值)

T3: D := D + 15 (读入数据库元素D的值，加上15后，再写回D的值)

除了这四个事务外，系统中无其他事务执行。设四个事务开始前，数据库元素A、B、C、D的值分别为A = 50，B = 30，C = 35，D = 15。在执行这四个事务的过程中，系统发生了故障。系统重启后，经故障恢复，数据库元素A、B、C、D的值被恢复为A = 50，B = 20，C = 70，D = 15。故障恢复时，数据库系统日志文件中包含如下12条日志记录，这里只给出部分日志记录。已知该数据库管理系统使用基于undo-redo日志的故障恢复技术，这段日志中仅有1个检查点

1	**<T0, start>**
2	<T0, A, 50, 70>
3	**<T2, start>**
4	<checkpoint (T0, T2)>
5	<T1, start>
6	**<T1, B, 30, 20>**
7	<T1, commit>
8	<T2, C, 35, 70>
9	**<T3, start>**
10	<T3, D, 15, 30>
11	**<T2, commit>**

请根据上述信息，回答下列问题：

1. 将日志文件补充完整，

2. 在故障恢复过程中，哪些事务需要redo，哪些事务需要undo？

目录

- 数据库恢复的必要性
- 使用日志的数据库恢复技术
- 使用检查点的数据库恢复技术
- 恢复算法
- **缓冲技术**

缓冲技术

- 日志缓冲技术
- 数据库缓冲技术

缓冲技术

- 日志缓冲技术
  - 通常一个日志记录远远小于存储器的读写单位。这样，经常向存储器写单个日志记录将导致很大的系统开销。
  - 对日志记录的读写操作使用缓冲技术，即在主存中设立缓冲区，其大小等于永久存储器的读写单位，被写的日志记录先存储到缓冲区，缓冲区满之后再一起永久写入存储器。

缓冲技术

- 日志缓冲技术
  - 缓冲区中的日志记录在系统发生故障时会丢失。为了保证事务的原子性，需要在数据库恢复协议上增加如下的规则：
    - (1) 任何事务T必须在日志记录<T, commit>已经永久写入存储器以后才可以进入提交状态。
    - (2) 任何事务T的非<T, commit>型日志记录必须在日志记录<T, commit>之前永久写入存储器。
    - (3) 主存缓冲区中的数据库数据必须在所有与这些数据有关的日志记录被永久写入存储器之后，才可以永久写入存储器中的数据库。

缓冲技术

- 数据库缓冲技术
  - 当系统要把数据块B_2读入主存并覆盖数据块B_1时，它必须按如下方式进行：
    - IF B_1已经被修改
    - THEN
    - 输出有关B_1中数据的所有日志永久地记录到存储器；
    - 输出B_1到磁盘；
    - 从磁盘输入B_2到内存缓冲区；
    - ENDIF。

总结

- 本章重点
  - 掌握故障的分类
  - 掌握使用日志的数据库恢复技术

Game Over

Wish you enjoy the course