数据库技术与应用

数据库恢复技术

第十章 数据库恢复技术

例子

- 银行转账：把10000元从一个账户甲转给另一个账户乙。

读账户甲的余额BALANCE；

BALANCE=BALANCE – 10000；

写回BALANCE；

读账户乙的余额BALANCE1；

BALANCE1 = BALANCE1 + 10000；

写回BALANCE1；

一、事务(Transaction)

- 定义
  - 一个数据库操作序列
  - 一个不可分割的工作单位（要么全做，要么不做）
  - 恢复和并发控制的基本单位

- 事务和程序比较
  - 在关系数据库中，一个事务可以是一条或多条SQL语句,也可以包含一个或多个程序。
  - 一个程序通常包含多个事务

定义事务

- 显式定义方式

```
BEGIN TRANSACTION
    SQL 语句1
    SQL 语句2
    。 。 。 。 。
COMMIT
```

```
BEGIN TRANSACTION
    SQL 语句1
    SQL 语句2
    。 。 。 。 。
ROLLBACK
```

- 隐式方式

  当用户没有显式地定义事务时，

  DBMS按缺省规定自动划分事务

二、事务的特性(ACID特性)

- 原子性（Atomicity）
  - 事务中的所有操作要么全部执行，要么都不执行。
- 一致性（Consistency）
  - 如果在执行事务之前数据库是一致的，那么在执行事务之后数据库也还应该是一致的。
- 隔离性（Isolation）
  - 即使多个事务并发执行，每个事务都感觉不到系统中有其他事务在执行，以保证数据库的一致性。
- 持续性（Durability ）
  - 事务成功执行后它对数据库的修改是永久的，即使系统出现故障。

第十章 数据库恢复技术

10.2 数据库恢复概述

- 故障是不可避免的
  - 系统故障：计算机软、硬件故障
  - 人为故障：操作员的失误、恶意的破坏等。
- 数据库的恢复

  把数据库从错误状态恢复到某一已知的正确状态(亦称为一致状态或完整状态)

  恢复子系统是DBMS的一个重要组成部分，而且相当庞大

故障的种类

- 事务内部的故障
- 系统故障
- 介质故障
- 计算机病毒

事务内部的故障

- 例子：银行转账事务，把一笔金额从一个账户甲转给另一个账户乙。

```
BEGIN TRANSACTION
读账户甲的余额BALANCE;
BALANCE=BALANCE-AMOUNT;  (AMOUNT 为转账金额)
写回BALANCE;
IF(BALANCE < 0 ) THEN
{
    打印'金额不足，不能转账';
    ROLLBACK;  (撤销刚才的修改，恢复事务)
}
ELSE
{
    读账户乙的余额BALANCE1;
    BALANCE1=BALANCE1+AMOUNT;
    写回BALANCE1;
    COMMIT;
}
```

事务内部的故障（续）

- 事务内部更多的故障是非预期的，不能由应用程序处理
  - 运算溢出
  - 并发事务发生死锁
  - 违反了某些完整性限制等

  以后，事务故障仅指这类非预期的故障
- 事务故障的恢复：撤消事务（UNDO）

二、系统故障

- 系统故障

  称为软故障，是指造成系统停止运转的任何事件，使得系统要重新启动。 如：

  - 特定类型的硬件错误（如CPU故障）
  - 操作系统故障
  - DBMS代码错误
  - 系统断电

- 后果
  - 整个系统的正常运行突然被破坏
  - 所有正在运行的事务都非正常终止
  - 内存中数据库缓冲区的信息全部丢失
  - 不破坏数据库

系统故障的恢复

- 发生系统故障时，事务未提交
  - 恢复策略：强行撤消（UNDO）所有未完成事务
- 发生系统故障时，事务已提交，但缓冲区中的信息尚未完全写回到磁盘上。
  - 恢复策略：重做（REDO）所有已提交的事务

三、介质故障

- 介质故障

  称为硬故障，指外存故障

  - 磁盘损坏
  - 磁头碰撞
  - 操作系统的某种潜在错误
  - 瞬时强磁场干扰

介质故障的恢复

- 装入数据库发生介质故障前某个时刻的数据副本
- 重做自此时始的所有成功事务，将这些事务已提交的结果重新记入数据库

四、计算机病毒

- 计算机病毒
  - 一种人为的故障或破坏，是一些恶作剧者研制的一种计算机程序
  - 可以繁殖和传播
- 危害
  - 破坏、盗窃系统中的数据
  - 破坏系统文件

故障小结

- 各类故障，对数据库的影响有两种可能性
  - 一是数据库本身被破坏
  - 二是数据库没有被破坏，但数据可能不正确，这是由于事务的运行被非正常终止造成的。

第十章 数据库恢复技术

10.4 恢复的实现技术

- 恢复操作的基本原理：冗余

  利用存储在系统其它地方的冗余数据来重建数据库中已被破坏或不正确的那部分数据

- 恢复机制涉及的关键问题

  1. 如何建立冗余数据
     - 数据转储（backup）
     - 登录日志文件（logging）
  2. 如何利用这些冗余数据实施数据库恢复

一、什么是数据转储

- 转储是指DBA将整个数据库复制到磁带或另一个磁盘上保存起来的过程，备用的数据称为后备副本或后援副本
- 如何使用
  - 数据库遭到破坏后可以将后备副本重新装入
  - 重装后备副本只能将数据库恢复到转储时的状态

静态转储

- 在系统中无运行事务时进行的转储操作
- 转储开始时数据库处于一致性状态
- 转储期间不允许对数据库的任何存取、修改活动
- 优点
  - 实现简单
  - 得到的一定是一个数据一致性的副本
- 缺点：降低了数据库的可用性
  - 转储必须等待正运行的用户事务结束
  - 新的事务必须等转储结束

动态转储

- 转储操作与用户事务并发进行
- 转储期间允许对数据库进行存取或修改
- 优点
  - 不用等待正在运行的用户事务结束
  - 不会影响新事务的运行
- 动态转储的缺点
  - 不能保证副本中的数据正确有效

[例]在转储期间的某个时刻*Tc*，系统把数据A=100转储到磁带上，而在下一时刻*Td*，某一事务将A改为200。转储结束后，后备副本上的A已是过时的数据了

动态转储

- 利用动态转储得到的副本进行故障恢复
  - 需要把动态转储期间各事务对数据库的修改活动登记下来，建立日志文件
  - 后备副本加上日志文件才能把数据库恢复到某一时刻的正确状态

2．海量转储与增量转储

- 海量转储（完全转储）：每次转储全部数据库
- 增量转储: 只转储上次转储后更新过的数据
- 海量转储与增量转储比较
  - 从恢复角度看，使用海量转储得到的后备副本进行恢复往往更方便
  - 但如果数据库很大，事务处理又十分频繁，则增量转储方式更实用更有效

10.4 恢复的实现技术

10.4.1 数据转储

10.4.2 登记日志文件

日志的作用

- 银行系统例子
  - 事务T：将50元从账户A转到账户B，A和B的初始值分别为1000和2000

read(A)

A = A-50

write(A)

read(B)

B = B+50

write(B)

假设故障发生在A新值写回磁盘之后，B新值写回磁盘之前，如何恢复？

1）重新执行T

2）不重新执行T

一、日志文件的格式和内容

- 什么是日志文件

  日志文件(log)是用来记录事务对数据库的更新操作的文件

- 日志文件的格式
  - 以记录为单位的日志文件
  - 以数据块为单位的日志文件

日志文件的格式和内容（续）

- 以记录为单位的日志文件内容
  - 各个事务的开始标记(BEGIN TRANSACTION)
  - 各个事务的结束标记(COMMIT或ROLLBACK)
  - 各个事务的所有更新操作
    - 事务标识（标明是哪个事务）
    - 操作类型（插入、删除或修改）
    - 操作对象（记录内部标识）
    - 更新前数据的旧值（对插入操作而言，此项为空值）
    - 更新后数据的新值（对删除操作而言，此项为空值）

以上均作为日志文件中的一个日志记录 (log record)

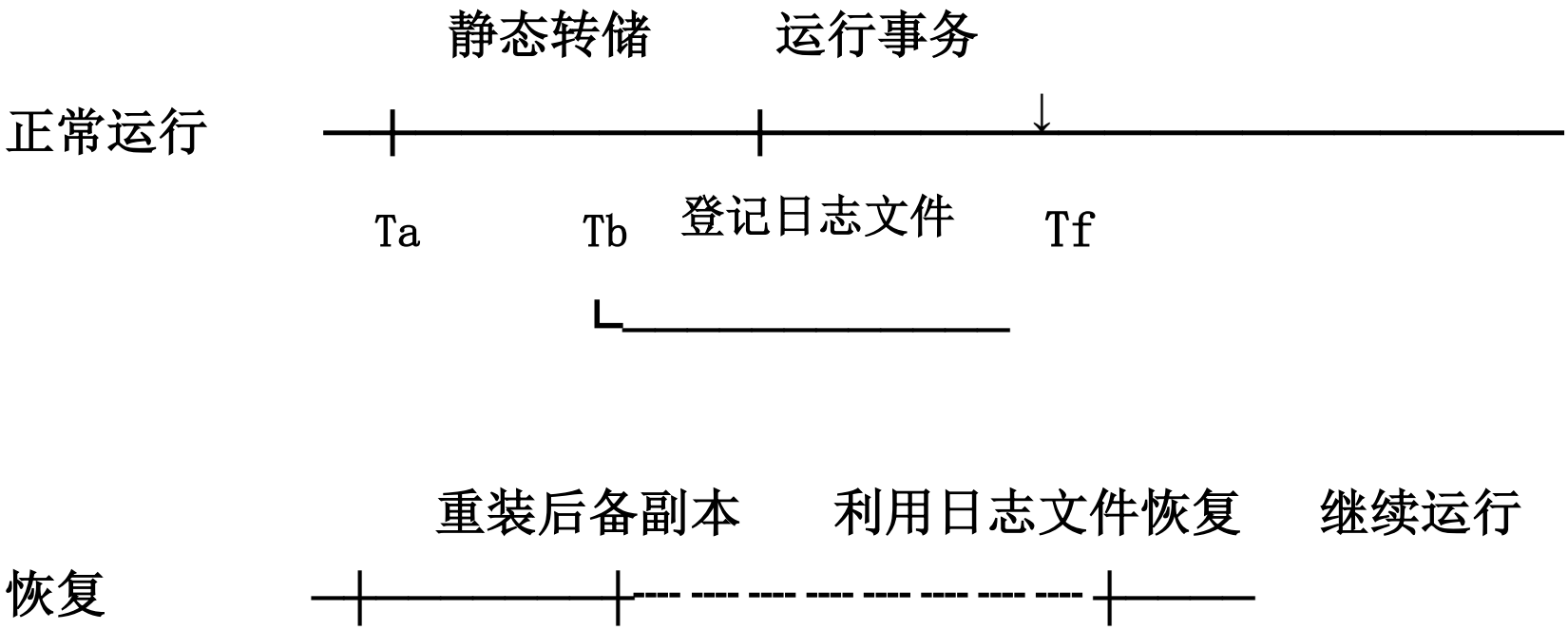
日志文件的格式和内容（续）

- 以数据块为单位的日志文件，每条日志记录的内容
  - 事务标识（标明是那个事务）
  - 被更新的数据块

二、日志文件的作用

- 进行事务故障恢复
- 进行系统故障恢复
- 协助后备副本进行介质故障恢复

利用静态转储副本和日志文件进行恢复



利用静态转储副本和日志文件进行恢复

上图中：

- 系统在T_a时刻停止运行事务，进行数据库转储
- 在T_b时刻转储完毕，得到T_b时刻的数据库一致性副本
- 系统运行到T_f时刻发生故障
- 为恢复数据库，首先由DBA重装数据库后备副本，将数据库恢复至T_b时刻的状态
- 重新运行自T_b～T_f时刻的所有更新事务，把数据库恢复到故障发生前的一致状态

三、登记日志文件

- 基本原则
  - 登记的次序严格按并行事务执行的时间次序
  - 必须先写日志文件，后写数据库
    - 写日志文件操作：把表示这个修改的日志记录写到日志文件
    - 写数据库操作：把对数据的修改写到数据库中

登记日志文件（续）

- 为什么要先写日志文件
  - 写数据库和写日志文件是两个不同的操作，在这两个操作之间可能发生故障
  - 如果先写了数据库修改，而在日志文件中没有登记下这个修改，则以后就无法恢复这个修改了
  - 如果先写日志，但没有修改数据库，按日志文件恢复时只不过是多执行一次不必要的UNDO操作，并不会影响数据库的正确性

第十章 数据库恢复技术

10.5 恢复策略

10.5.1 事务故障的恢复

10.5.2 系统故障的恢复

10.5.3 介质故障的恢复

10.5.1 事务故障的恢复

- 事务故障：事务在运行至正常终止点前被终止
- 恢复方法
  - 由恢复子系统应利用日志文件撤消（UNDO）此事务已对数据库进行的修改
- 事务故障的恢复由系统自动完成，对用户是透明的，不需要用户干预

事务故障的恢复步骤

1. 反向扫描文件日志（即从最后向前扫描日志文件），查找该事务的更新操作。
2. 对该事务的更新操作执行逆操作。即将日志记录中“更新前的值”写入数据库。
   - 插入操作，“更新前的值”为空，则相当于做删除操作
   - 删除操作，“更新后的值”为空，则相当于做插入操作
   - 若是修改操作，则相当于用修改前值代替修改后值
3. 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。
4. 如此处理下去，直至读到此事务的开始标记，事务故障恢复就完成了。

10.5 恢复策略

10.5.1 事务故障的恢复

10.5.2 系统故障的恢复

10.5.3 介质故障的恢复

10.5.2 系统故障的恢复

- 系统故障造成数据库不一致状态的原因
  - 未完成事务对数据库的更新已写入数据库
  - 已提交事务对数据库的更新还留在缓冲区没来得及写入数据库
- 恢复方法
  - 1. Undo 故障发生时未完成的事务
  - 2. Redo 已完成的事务
- 系统故障的恢复由系统在重新启动时自动完成，不需要用户干预

系统故障的恢复步骤

1. 正向扫描日志文件（即从头扫描日志文件）

- 重做(REDO) 队列: 在故障发生前已经提交的事务
  - ➢这些事务既有BEGIN TRANSACTION记录，也有COMMIT记录
- 撤销 (Undo)队列:故障发生时尚未完成的事务
  - ➢ 这些事务只有BEGIN TRANSACTION记录，无相应的COMMIT记录

系统故障的恢复步骤

2. 对撤销(Undo)队列事务进行撤销(UNDO)处理
   - 反向扫描日志文件，对每个UNDO事务的更新操作执行逆操作
   - 即将日志记录中“更新前的值”写入数据库
3. 对重做(Redo)队列事务进行重做(REDO)处理
   - 正向扫描日志文件，对每个REDO事务重新执行登记的操作
   - 即将日志记录中“更新后的值”写入数据库

系统故障恢复举例

- 事务T0：将50元从账户A转到账户B，执行前A、B的值为1000、2000

read(A)

A = A-50

write(A)

read(B)

B = B+50

write(B)

- 事务T1：从账户C取出100元，执行前C的值为700

read(C)

C = C- 100

write(C)

10.5 恢复策略

10.5.1 事务故障的恢复

10.5.2 系统故障的恢复

10.5.3 介质故障的恢复

10.5.3 介质故障的恢复

1.重装数据库

2.重做已完成的事务

介质故障的恢复（续）

- 恢复步骤

1. 装入最新的后备数据库副本(离故障发生时刻最近的转储副本)，使数据库恢复到最近一次转储时的一致性状态。
   - 对于静态转储的数据库副本，装入后数据库即处于一致性状态
   - 对于动态转储的数据库副本，还须同时装入转储时刻的日志文件副本，利用与恢复系统故障的方法（即REDO+UNDO），才能将数据库恢复到一致性状态。

介质故障的恢复（续）

2. 装入有关的日志文件副本(转储结束时刻的日志文件副本)，重做已完成的事务。

- 首先扫描日志文件，找出故障发生时已提交的事务的标识，将其记入重做队列。
- 然后正向扫描日志文件，对重做队列中的所有事务进行重做处理。即将日志记录中“更新后的值”写入数据库。

介质故障的恢复（续）

介质故障的恢复需要DBA介入

- DBA的工作
  - 重装最近转储的数据库副本和有关的各日志文件副本
  - 执行系统提供的恢复命令
- 具体的恢复操作仍由DBMS完成

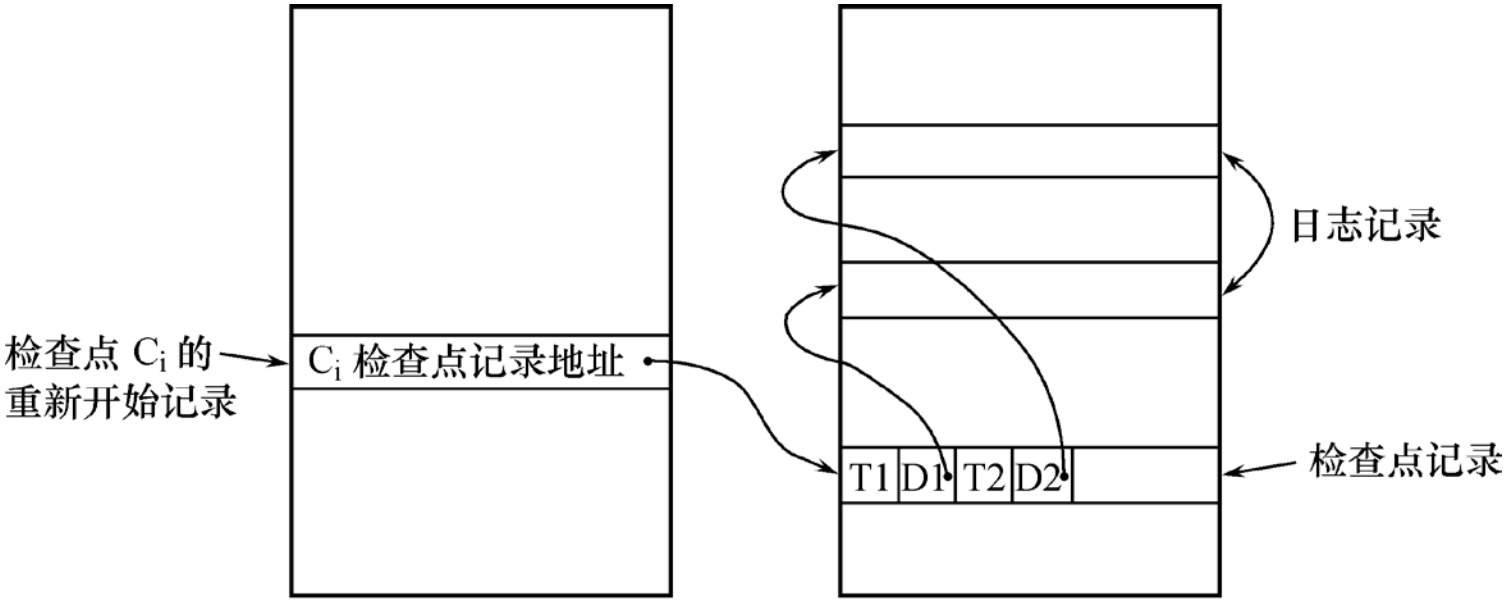
第十章 数据库恢复技术

一、问题的提出

- 两个问题
  - 搜索整个日志将耗费大量的时间
  - REDO处理：重新执行，浪费了大量时间
- 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）
  - 增加重新开始文件
  - 恢复子系统在登录日志文件期间动态地维护日志

二、检查点技术

- 检查点记录的内容
  - 1. 建立检查点时刻所有正在执行的事务清单
  - 2. 这些事务最近一个日志记录的地址
- 重新开始文件的内容
  - 记录各个检查点记录在日志文件中的地址



动态维护日志文件的方法

- 动态维护日志文件的方法

  周期性地执行如下操作：建立检查点，保存数据库状态。

  具体步骤是：

  - 1.将当前日志缓冲区中的所有日志记录写入磁盘的日志文件上
  - 2.在日志文件中写入一个检查点记录
  - 3.将当前数据缓冲区的所有数据记录写入磁盘的数据库中
  - 4.把检查点记录在日志文件中的地址写入一个重新开始文件

建立检查点

- 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

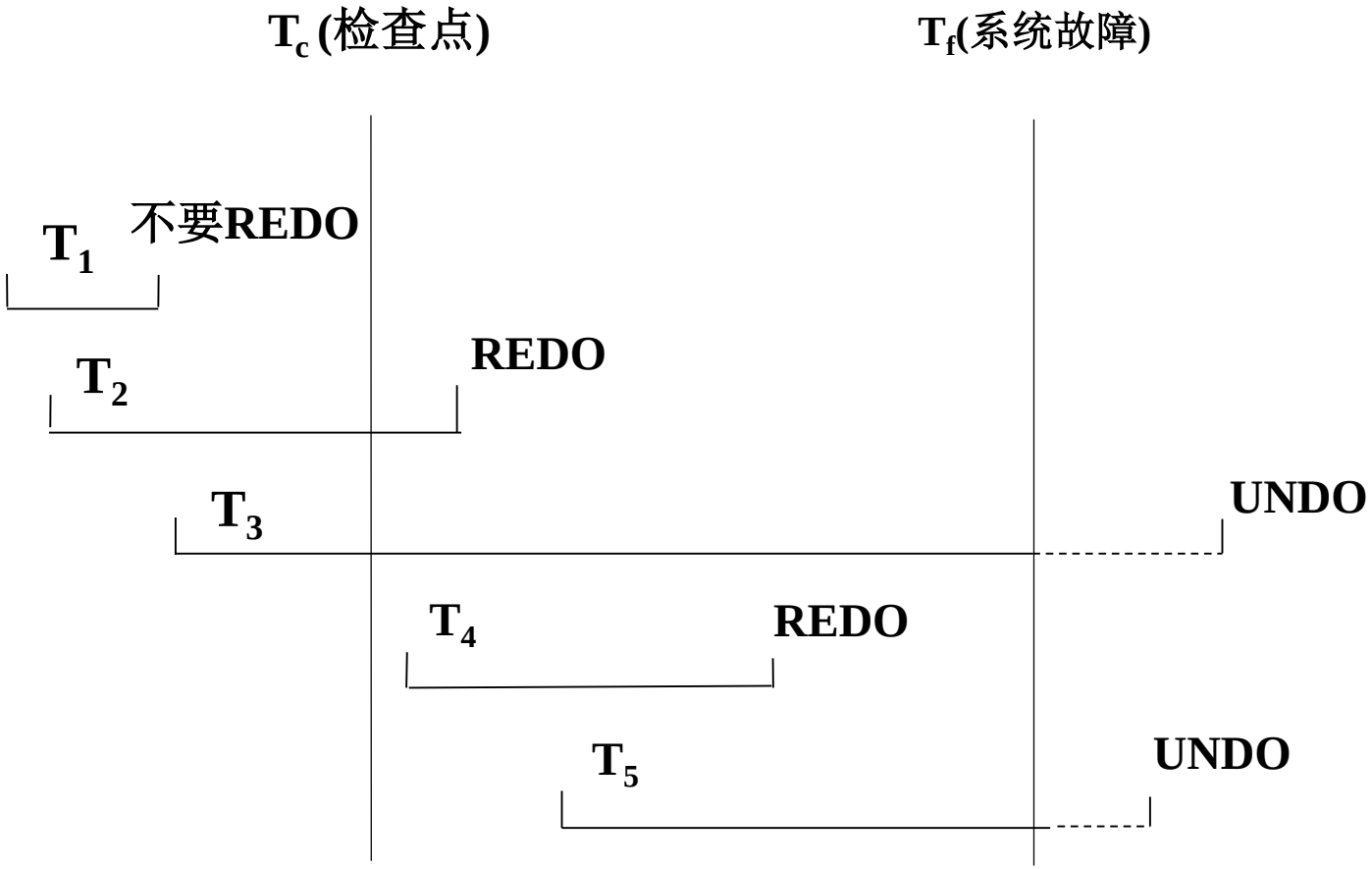
三、利用检查点的恢复策略

- 使用检查点方法可以改善恢复效率
  - 当事务T在一个检查点之前提交

    T对数据库所做的修改已写入数据库
  - 写入时间是在这个检查点建立之前或在这个检查点建立之时
  - 在进行恢复处理时，没有必要对事务T执行REDO操作

利用检查点的恢复策略（续）

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



利用检查点的恢复策略（续）

- 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

利用检查点的恢复步骤

1. 从重新开始文件中找到最后一个检查点记录在日志文件中的地址，由该地址在日志文件中找到最后一个检查点记录

2. 由该检查点记录得到检查点建立时刻所有正在执行的事务清单ACTIVE-LIST

- 建立两个事务队列
  - ❿ UNDO-LIST
  - ❿ REDO-LIST
- 把ACTIVE-LIST暂时放入UNDO-LIST队列，REDO队列暂为空。

利用检查点的恢复策略（续）

3. 从检查点开始正向扫描日志文件，直到日志文件结束
   - 如有新开始的事务T_i，把T_i暂时放入UNDO-LIST队列
   - 如有提交的事务T_j，把T_j从UNDO-LIST队列移到REDO-LIST队列
4. 对UNDO-LIST中的每个事务执行UNDO操作

   对REDO-LIST中的每个事务执行REDO操作

第十章 数据库恢复技术

10.7 数据库镜像

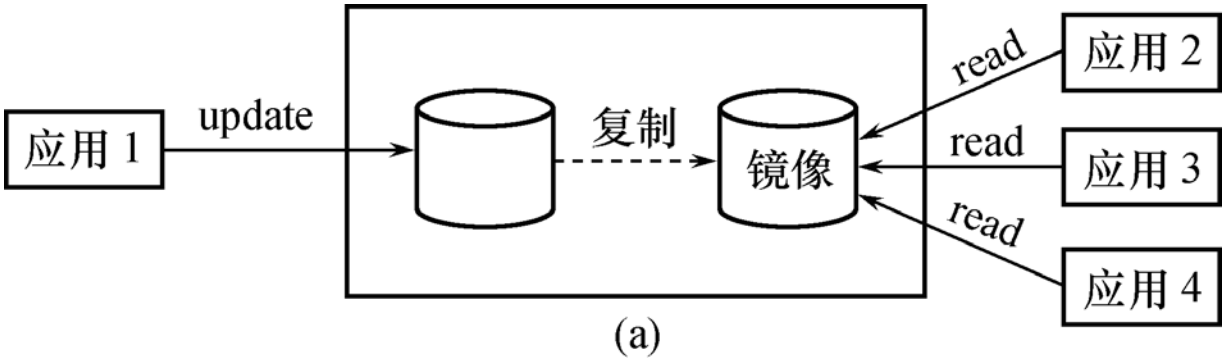
- 介质故障是对系统影响最为严重的一种故障，严重影响数据库的可用性
  - 介质故障恢复比较费时
  - 为预防介质故障，DBA必须周期性地转储数据库

- 提高数据库可用性的解决方案
  - 数据库镜像（Mirror）

数据库镜像（续）

- 数据库镜像
  - DBMS自动把整个数据库或其中的关键数据复制到另一个磁盘上
  - DBMS自动保证镜像数据与主数据库的一致性

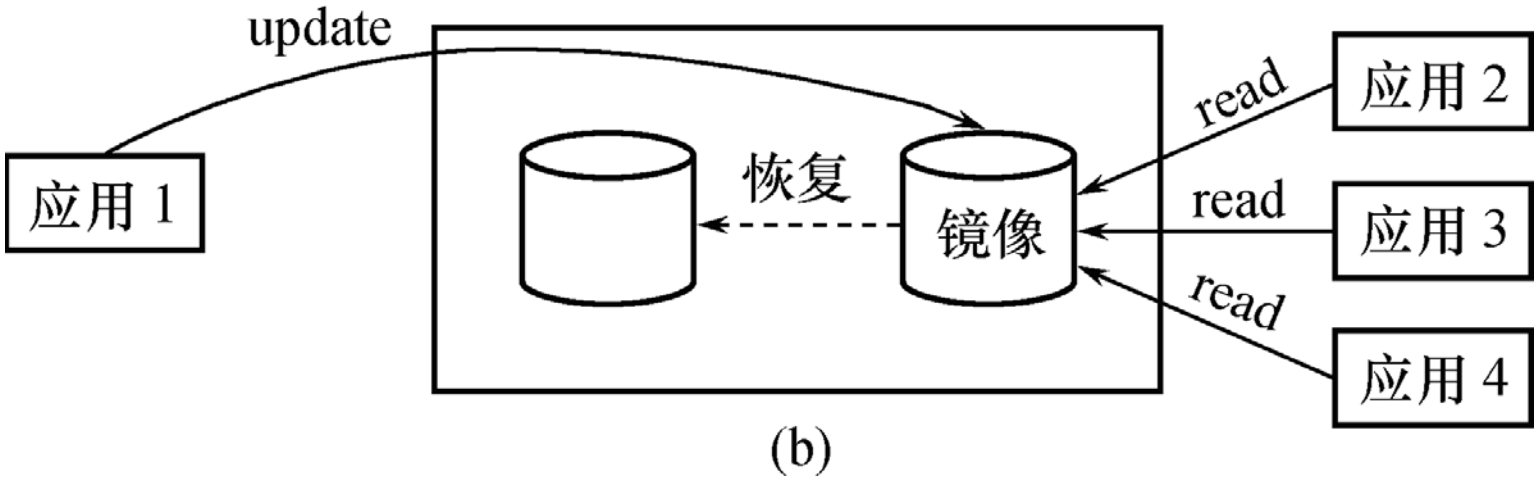
每当主数据库更新时，DBMS自动把更新后的数据复制过去（如下图所示）



(a)

数据库镜像的用途

- 出现介质故障时
  - 可由镜像磁盘继续提供使用
  - 同时DBMS自动利用镜像磁盘数据进行数据库的恢复
  - 不需要关闭系统和重装数据库副本



(b)

数据库镜像（续）

- 没有出现故障时
  - 可用于并发操作
  - 一个用户对数据加排他锁修改数据，其他用户可以读镜像数据库上的数据，而不必等待该用户释放锁

- **频繁地复制数据自然会降低系统运行效率**

- **在实际应用中用户往往只选择对关键数据和日志文件镜像，而不是对整个数据库进行镜像**

第十章 数据库恢复技术

10.8 小结

- 如果数据库只包含成功事务提交的结果，就说数据库处于一致性状态。保证数据一致性是对数据库的最基本的要求。
- 事务是数据库的逻辑工作单位
  - DBMS保证系统中一切事务的原子性、一致性、隔离性和持续性

小结（续）

- 常用恢复技术
  - 事务故障的恢复
    - UNDO
  - 系统故障的恢复
    - UNDO + REDO
  - 介质故障的恢复
    - 重装备份并恢复到一致性状态 + REDO

数据库系统概论

An Introduction to Database System

第十一章 并发控制

问题的产生

- 多用户数据库系统的存在

  允许多个用户同时使用的数据库系统

  - 飞机定票数据库系统
  - 银行数据库系统

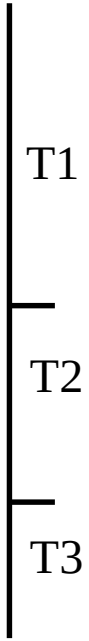
  特点：在同一时刻并发运行的事务数可达数百个

问题的产生（续）

- 不同的多事务执行方式

(1)事务串行执行

- 每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行
- 不能充分利用系统资源，发挥数据库共享资源的特点

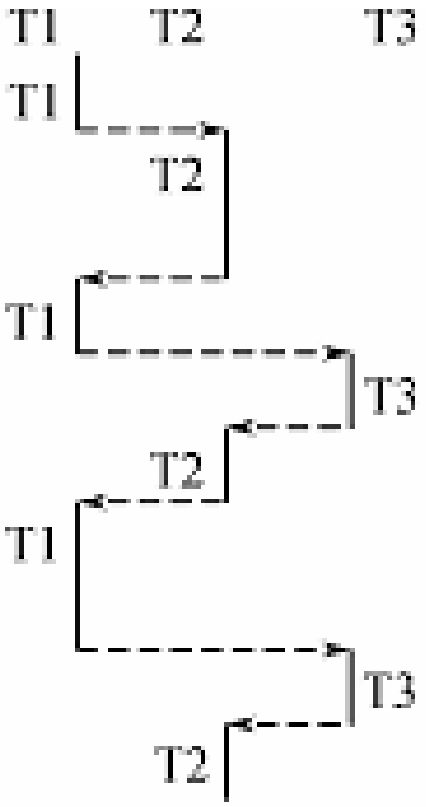


事务的串行执行方式

问题的产生（续）

(2) 交叉并发方式（Interleaved Concurrency）

- 在单处理机系统中，事务的并行执行是这些并行事务的并行操作轮流交叉运行
- 单处理机系统中的并行事务并没有真正地并行运行，但能够减少处理机的空闲时间，提高系统的效率



问题的产生（续）

(3)同时并发方式（simultaneous concurrency）

- 多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务真正的并行运行

问题的产生（续）

- 事务并发执行带来的问题
  - 会产生多个事务同时存取同一数据的情况
  - 可能会存取和存储不正确的数据，破坏事务一致性和数据库的一致性

11.1 并发控制概述

- 并发控制机制的任务
  - 对并发操作进行正确调度
  - 保证事务的隔离性
  - 保证数据库的一致性

并发控制概述（续）

并发操作带来数据的不一致性实例

[例1]飞机订票系统中的一个活动序列

① 甲售票点(甲事务)读出某航班的机票余额A，设A=16；

② 乙售票点(乙事务)读出同一航班的机票余额A，也为16；

③ 甲售票点卖出一张机票，修改余额A←A-1，所以A为15，把A写回数据库；

④ 乙售票点也卖出一张机票，修改余额A←A-1，所以A为15，把A写回数据库

- 结果明明卖出两张机票，数据库中机票余额只减少1

并发控制概述（续）

- 这种情况称为数据库的不一致性，是由并发操作引起的。
- 在并发操作情况下，对甲、乙两个事务的操作序列的调度是随机的。
- 若按上面的调度序列执行，甲事务的修改就被丢失。
  - 原因：第4步中乙事务修改A并写回后覆盖了甲事务的修改

并发控制概述（续）

- 并发操作带来的数据不一致性
  - 丢失修改（Lost Update）
  - 不可重复读（Non-repeatable Read）
  - 读“脏”数据（Dirty Read）
- 记号
  - R(x):读数据x
  - W(x):写数据x

1. 丢失修改

- 两个事务T_1和T_2读入同一数据并修改，T_2的提交结果破坏了T_1提交的结果，导致T_1的修改被丢失。
- 上面飞机订票例子就属此类

丢失修改（续）

T_1	T_2
① R(A)=16	
②	R(A)=16
③ A←A-1 W(A)=15W	
④	A←A-1 W(A)=15

丢失修改

2. 不可重复读

- 不可重复读是指事务T_1读取数据后，事务T_2执行更新操作，使T_1无法再现前一次读取结果。

不可重复读（续）

- 不可重复读包括三种情况：

(1)事务T_1读取某一数据后，事务T_2对其做了修改，当事务T_1再次读该数据时，得到与前一次不同的值

不可重复读（续）

例如：

T_1	T_2
① R(A)=50	
R(B)=100	
求和=150	
②	R(B)=100
	B←B*2
	(B)=200
③ R(A)=50	
R(B)=200	
和=250	
(验算不对)	

不可重复读

- T1读取B=100进行运算
- T2读取同一数据B，对其进行修改后将B=200写回数据库。
- T1为了对读取值校对重读B，B已为200，与第一次读取值不一致

不可重复读（续）

(2)事务T1按一定条件从数据库中读取了某些数据记录后，事务T2删除了其中部分记录，当T1再次按相同条件读取数据时，发现某些记录消失了

(3)事务T1按一定条件从数据库中读取某些数据记录后，事务T2插入了一些记录，当T1再次按相同条件读取数据时，发现多了一些记录。

后两种不可重复读有时也称为幻影现象（Phantom Row）

3. 读“脏”数据

读“脏”数据是指：

- 事务T1修改某一数据，并将其写回磁盘
- 事务T2读取同一数据后，T1由于某种原因被撤销
- 这时T1已修改过的数据恢复原值，T2读到的数据就与数据库中的数据不一致
- T2读到的数据就为“脏”数据，即不正确的数据

读“脏”数据（续）

例如

T_1	T_2
① R(C)=100 C←C*2 W(C)=200	
②	R(C)=200
③ROLLBACK C恢复为100	

读“脏”数据

- T1将C值修改为200，T2读到C为200
- T1由于某种原因撤销，其修改作废，C恢复原值100
- 这时T2读到的C为200，与数据库内容不一致，就是“脏”数据

并发控制概述（续）

- 数据不一致性：由于并发操作破坏了事务的隔离性
- 并发控制就是要用正确的方式调度并发操作，使一个用户事务的执行不受其他事务的干扰，从而避免造成数据的不一致性

并发控制概述（续）

- 并发控制的主要技术
  - 封锁(Locking)
  - 时间戳(Timestamp)
  - 乐观控制法
- 商用的DBMS一般都采用封锁方法

什么是封锁

- 封锁就是事务T在对某个数据对象（例如表、记录等）操作之前，先向系统发出请求，对其加锁
- 加锁后事务T就对该数据对象有了一定的控制，在事务T释放它的锁之前，其它的事务不能更新此数据对象。

基本封锁类型

- 一个事务对某个数据对象加锁后究竟拥有什么样的控制由封锁的类型决定。
- 基本封锁类型
  - 排它锁（Exclusive Locks，简记为X锁）
  - 共享锁（Share Locks，简记为S锁）

排它锁

- 排它锁又称为写锁
- 若事务T对数据对象A加上X锁，则只允许T读取和修改A，其它任何事务都不能再对A加任何类型的锁，直到T释放A上的锁
- 保证其他事务在T释放A上的锁之前不能再读取和修改A

共享锁

- 共享锁又称为读锁
- 若事务T对数据对象A加上S锁，则其它事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁
- 保证其他事务可以读A，但在T释放A上的S锁之前不能对A做任何修改

锁的相容矩阵

T_2 \ T_1	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

Y=Yes，相容的请求
N=No，不相容的请求

使用封锁机制解决丢失修改问题

例：

T_1	T_2
① Xlock A	
② R(A)=16	
	Xlock A
③ A←A-1	等待
W(A)=15	等待
Commit	等待
Unlock A	等待
④	获得Xlock A
	R(A)=15
	A←A-1
⑤	W(A)=14
	Commit
	Unlock A

没有丢失修改

- 事务T1在读A进行修改之前先对A加X锁
- 当T2再请求对A加X锁时被拒绝
- T2只能等待T1释放A上的锁后T2获得对A的X锁
- 这时T2读到的A已经是T1更新过的值15
- T2按此新的A值进行运算，并将结果值A=14送回到磁盘。避免了丢失T1的更新。

使用封锁机制解决读“脏”数据问题

例

T_1	T_2
① Xlock C	
R(C)=100	
C←C*2	
W(C)=200	
②	Slock C
	等待
③ ROLLBACK	等待
(C恢复为100)	等待
Unlock C	等待
④	获得Slock C
	R(C)=100
⑤	Commit C
	Unlock C

不读“脏”数据

- 事务T1在对C进行修改之前，先对C加X锁，修改其值后写回磁盘
- T2请求在C上加S锁，因T1已在C上加了X锁，T2只能等待
- T1因某种原因被撤销，C恢复为原值100
- T1释放C上的X锁后T2获得C上的S锁，读C=100。避免了T2读“脏”数据

实验内容3：备份和恢复

- 用Transact-SQL完成以下操作
  - (1) 创建一个备份设备。
  - (2) 完整备份实验数据库到备份设备。
  - (3) 向实验数据库中某个表插入若干条记录。
  - (4)备份数据库事务日志到备份设备，并查看日志格式。
  - (5)利用第2步所得的完整备份，恢复到插入记录前的状态。
  - (6)利用第4步所得的事务日志，恢复到插入记录后的状态。
- 备份恢复命令
  - SQL Server：sp_addumpdevice、BACKUP DATABASE/LOG、RESTORE DATABASSE/LOG
  - MySQL：mysqldump、source、mysqlbinlog

提交

- 12月1日晚12:00前发到助教邮箱
  - 15651638081@163.com
- 命名格式
  - 班号-学号-姓名-实验N.doc/docx/rar