数据库恢复技术

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
  - 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

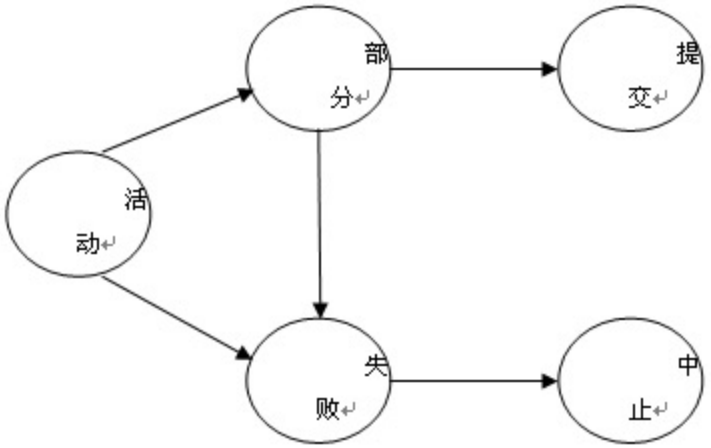
二、事务的特性(ACID特性)

事务的ACID特性：

- 原子性（Atomicity）
- 一致性（Consistency）
- 隔离性（Isolation）
- 持续性（Durability ）

事务状态

- 活动状态
- 部分提交状态
- 失败状态
- 提交状态
- 终止状态



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)

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

一、事务内部的故障

- 事务内部的故障
  - 有的是可以通过事务程序本身发现的(见下面转账事务的例子)
  - 有的是非预期的

事务内部的故障（续）

- 例如，银行转账事务，这个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）

- 这个例子所包括的两个更新操作要么全部完成要么全部不做。否则就会使数据库处于不一致状态，例如只把账户甲的余额减少了而没有把账户乙的余额增加。
- 在这段程序中若产生账户甲余额不足的情况，应用程序可以发现并让事务滚回，撤销已作的修改，恢复数据库到正确状态。

事务内部的故障（续）

- 事务内部更多的故障是非预期的，是不能由应用程序处理的。
  - 运算溢出
  - 并发事务发生死锁而被选中撤销该事务
  - 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。

  - 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
  - 不破坏数据库
  - 内存中数据库缓冲区的信息全部丢失

系统故障的常见原因

- 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

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。

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

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

二、转储方法

1．静态转储与动态转储

2．海量转储与增量转储

3．转储方法小结

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
- 得到的一定是一个数据一致性的副本
- 优点：实现简单
- 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T_c，系统把数据A=100转储到磁带上，而在下一时刻T_d，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: 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

3．转储方法小结

- 转储方法分类

		转储状态	
		动态转储	静态转储
转储方式	海量转储	动态海量转储	静态海量转储
	增量转储	动态增量转储	静态增量转储

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

  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）

- 以记录为单位的日志文件，每条日志记录的内容
  - 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）

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（续）

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

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

事务故障的恢复步骤

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。

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

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

检查点技术

- 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

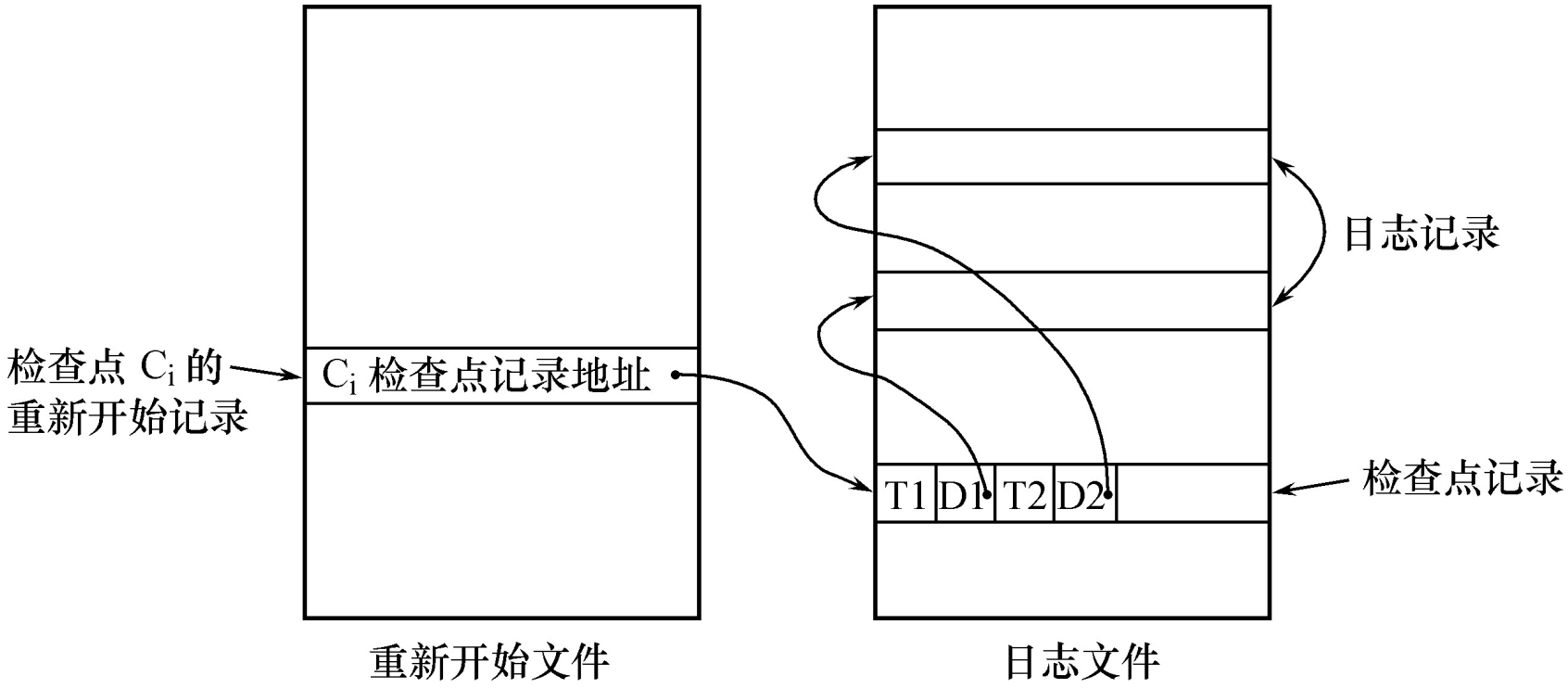
解决方案

- 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

检查点技术（续）



具有检查点的日志文件和重新开始文件

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

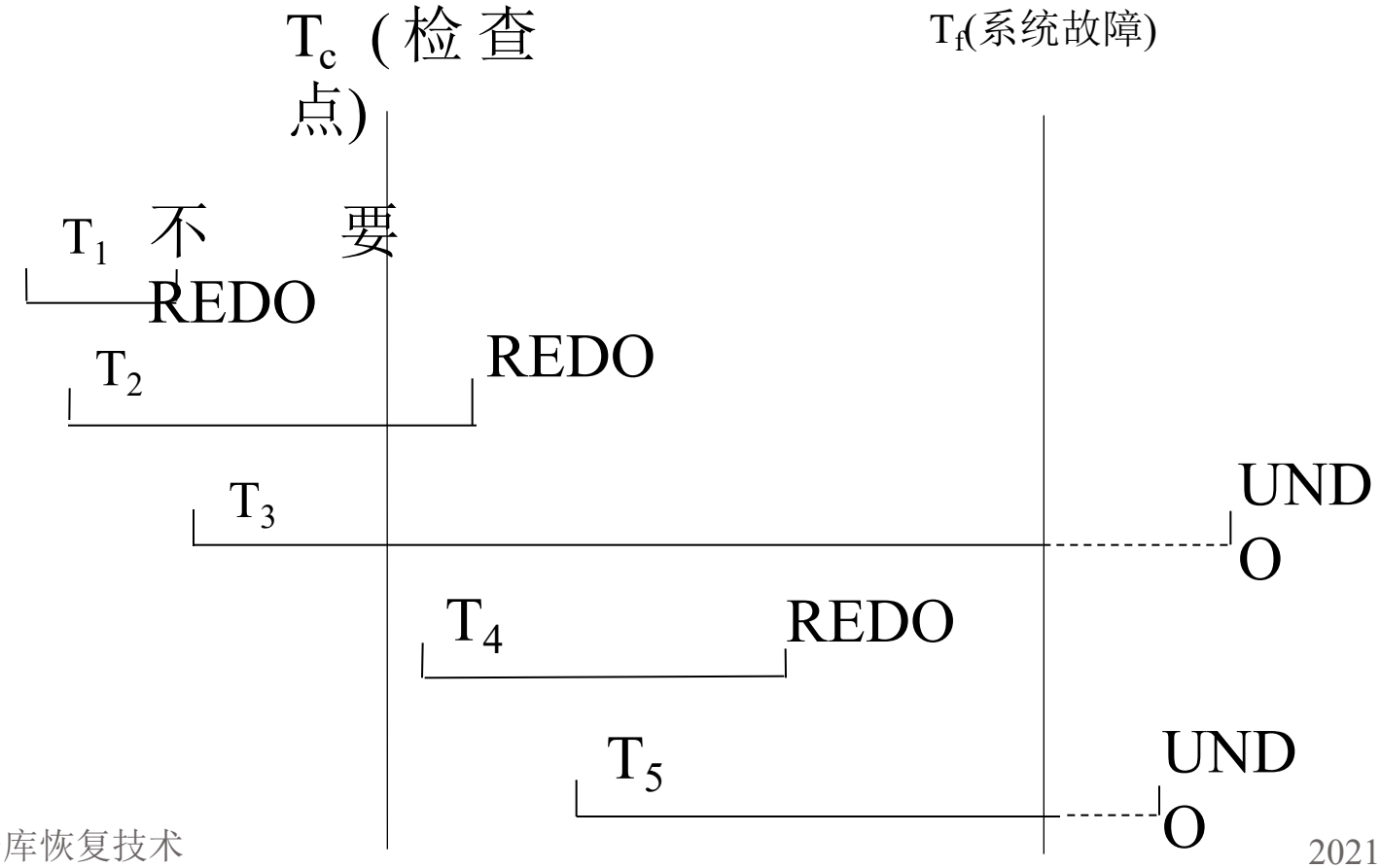
三、利用检查点的恢复策略

- 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

利用检查点的恢复策略（续）

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
  - UNDO-LIST
  - 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

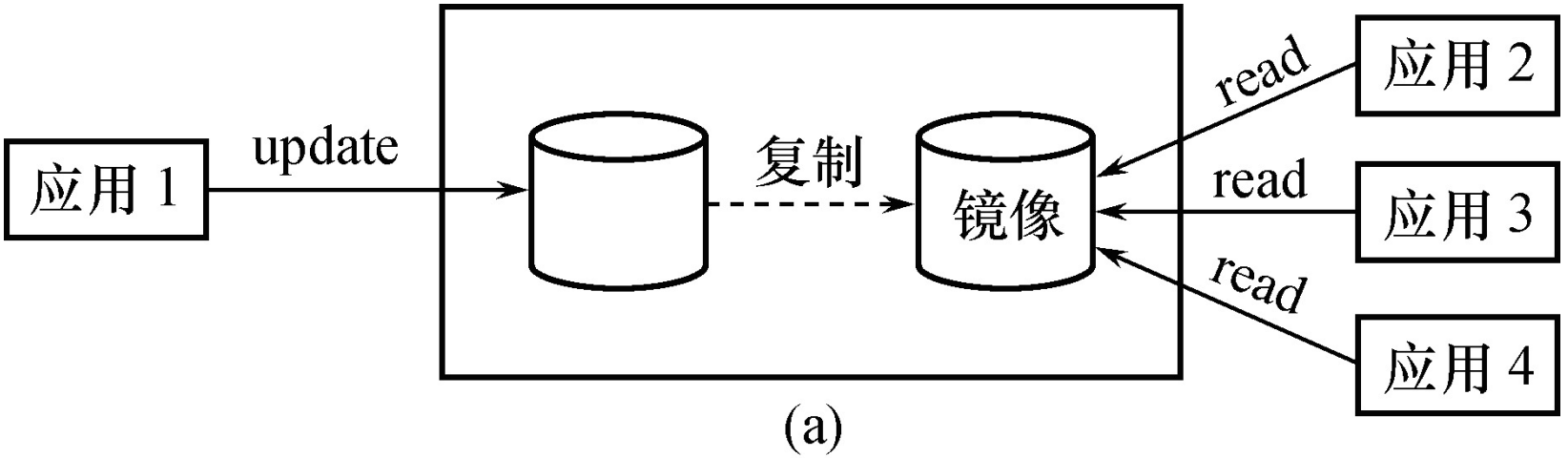
对REDO-LIST中的每个事务执行REDO操作

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）

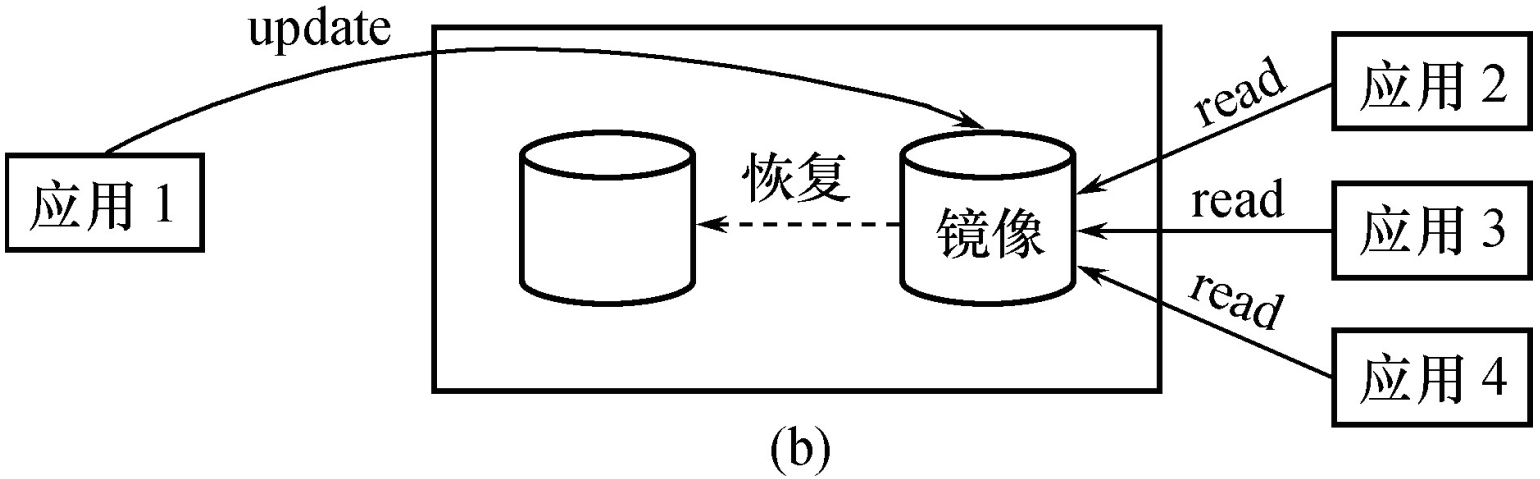
数据库镜像（续）



(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(如下图所示)



(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

数据库镜像（续）

- 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

1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）

- DBMS必须对事务故障、系统故障和介质故障进行恢复
- 恢复中最经常使用的技术：数据库转储和登记日志文件
- 恢复的基本原理：利用存储在后备副本、日志文件和数据库镜像中的冗余数据来重建数据库

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

小结（续）

- 提高恢复效率的技术
  - 检查点技术
    - 可以提高系统故障的恢复效率
    - 可以在一定程度上提高利用动态转储备份进行介质故障恢复的效率
  - 镜像技术
    - 镜像技术可以改善介质故障的恢复效率