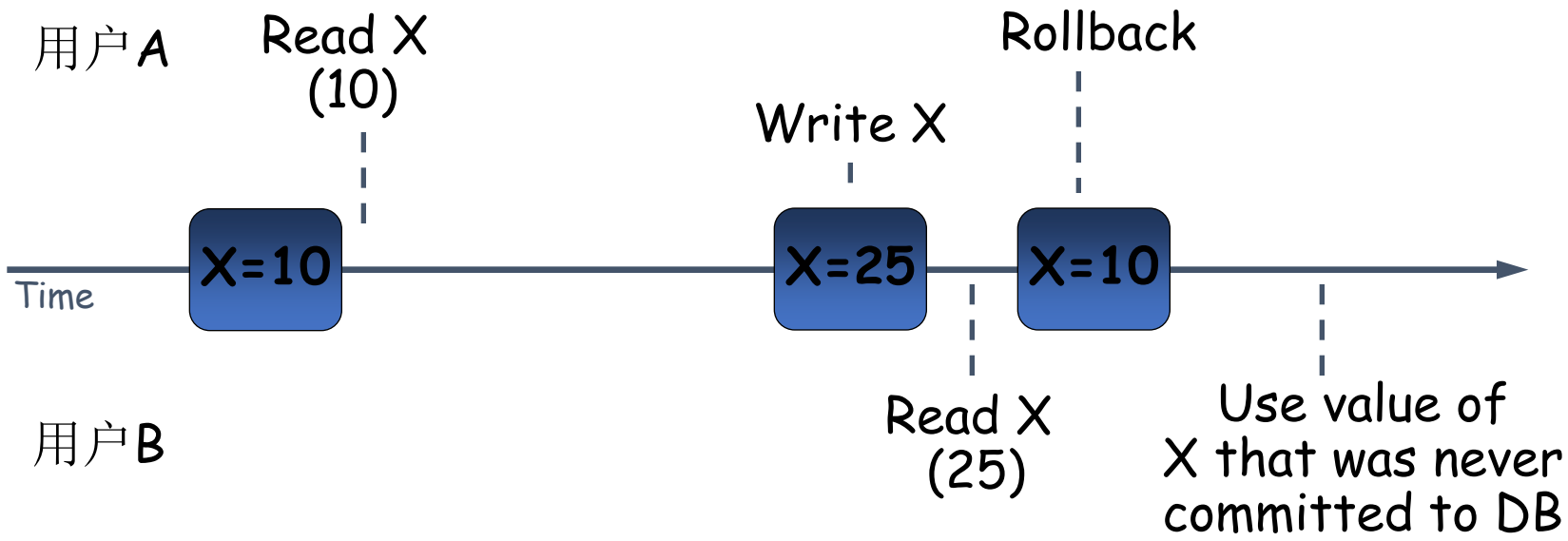
# 第十章 数据库恢复技术

# 内容提要

- 事务的基本概念
- 数据库恢复概述
- 故障种类
- 恢复策略
- 数据库的镜像

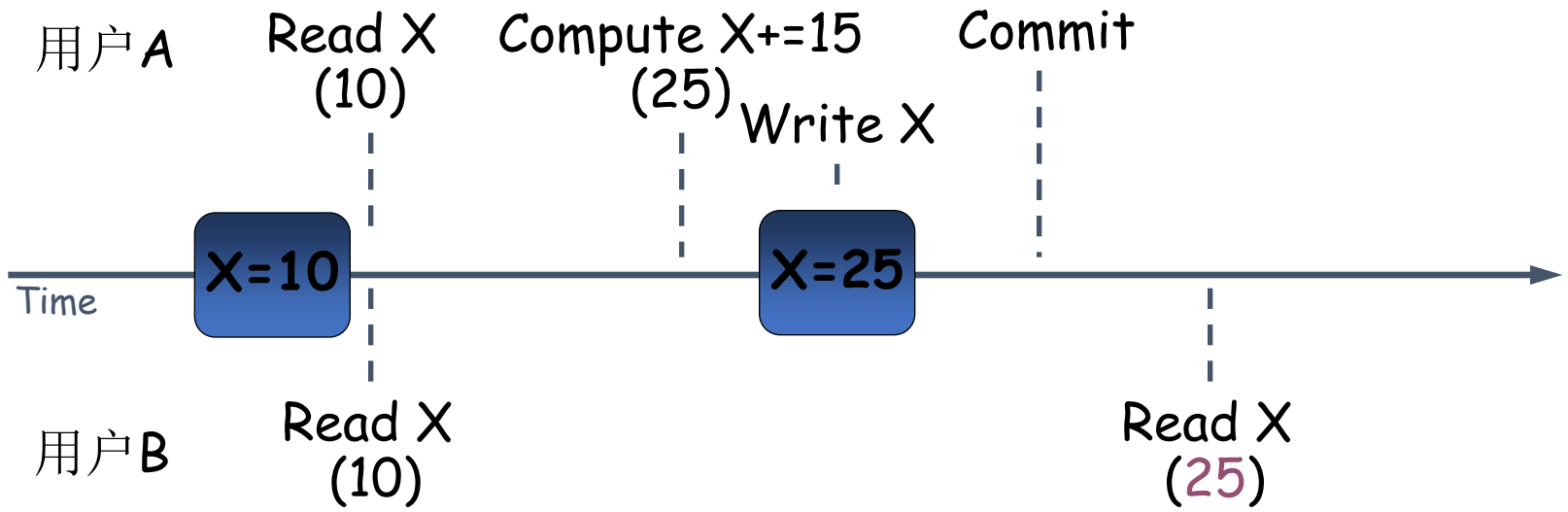
# 一般多用户系统存在的问题

- 用户B 读脏数据



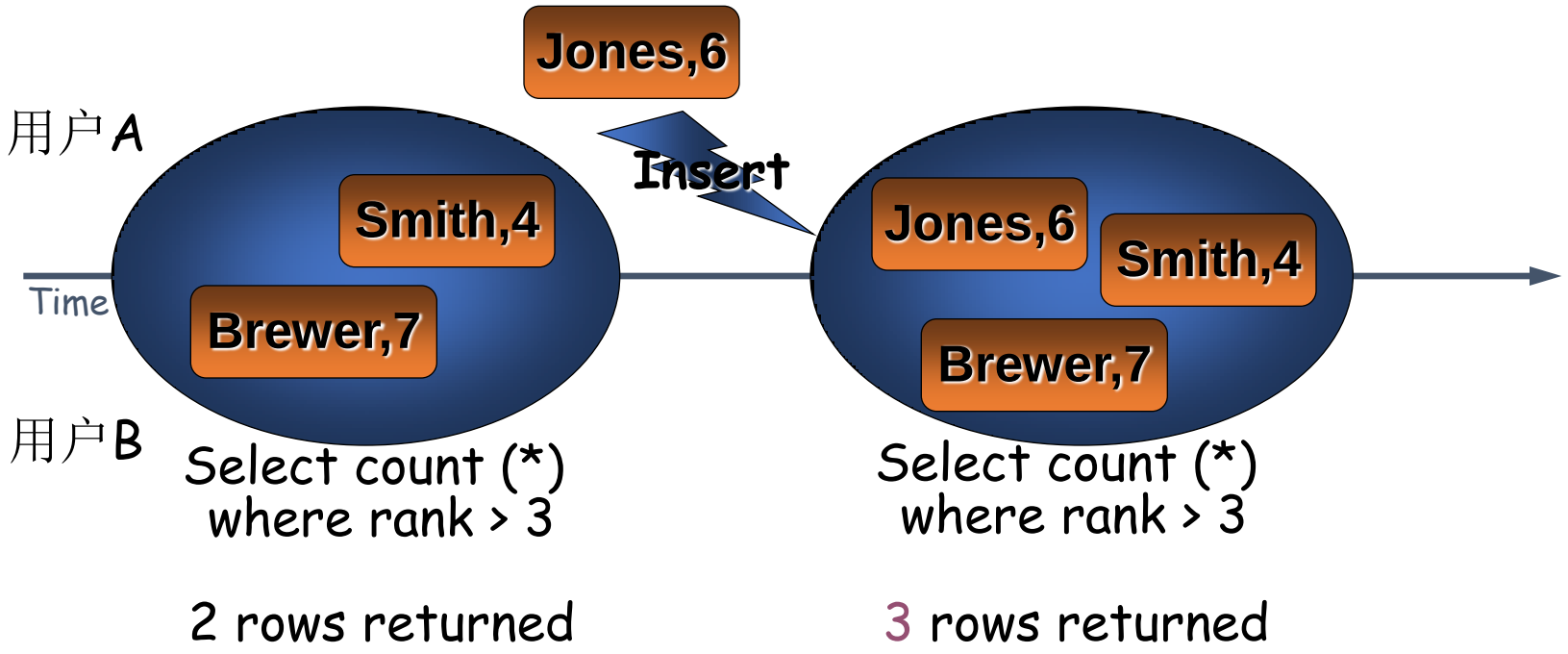
# 一般多用户系统存在的问题（续）

- 用户B 不可重复读



# 一般多用户系统存在的问题（续）

- 用户B出现幻影元组



# 分析问题

- 用户B 执行了一系列操作
- 除了用户B，还有其他用户在使用

- 如何保证用户B执行的**正确性**?
- 如何保证用户B和其他用户的**隔离性**?

→ 引入“事务”

# 事务

- 事务(Transaction)是用户定义的一个数据库**操作序列**，这些操作要么全做，要么全不做，是一个不可分割的工作单位。

- 事务和程序是两个概念
  - 在关系数据库中，一个事务可以是一条SQL语句，一组SQL语句或整个程序
  - 一个程序通常包含多个事务

- 事务是恢复和并发控制的基本单位

# 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
```

- 事务异常终
- 事务运行的
  能继续执行
- 系统将事务
  成的操作全部撤销
- 事务滚回到开始时的状态

- 事务正常结束
- 提交事务的所有操作（读+更新）
- 事务中所有对数据库的更新写回到磁盘上的物理数据库中

# 事务结束

- COMMIT
  - 事务正常结束
  - 提交事务的所有操作（读+更新）
  - 事务中所有对数据库的更新写回到磁盘上的物理数据库中
- ROLLBACK
  - 事务异常终止
  - 事务运行的过程中发生了故障，不能继续执行
  - 系统将事务中对数据库的所有已完成的操作全部撤销
  - 事务滚回到开始时的状态

# 事务特性——ACID

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

# 原子性

- 事务是数据库的逻辑工作单位
  - 事务中包括的诸操作要么都做，要么都不做

# 一致性

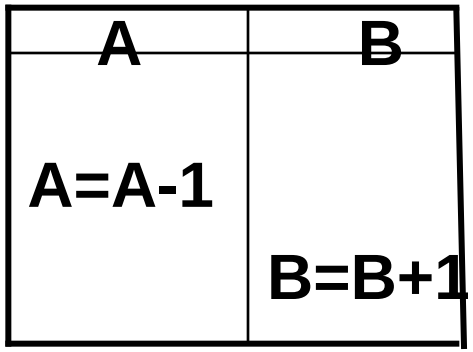
- 事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态
- 一致性状态
  - 数据库中只包含成功事务提交的结果
- 不一致状态
  - 数据库系统运行中发生故障，有些事务尚未完成就被迫中断;
  - 这些未完成事务对数据库所做的修改有一部分已写入物

    理数据库，这时数据库就处于一种不正确的状态

# 一致性与原子性

银行转帐：从帐号A中取出一万元，存入帐号B。

- 定义一个事务，该事务包括两个操作




- 这两个操作要么全做，要么全不做
  - 全做或者全不做，数据库都处于一致性状态。
  - 如果只做一个操作，用户逻辑上就会发生错误，少了一万元，数据库就处于不一致性状态。

# 隔离性

一个事务的执行不能被其他事务干扰

- 一个事务内部的操作及使用的数据对其他并发事务是隔离的
- 并发执行的各个事务之间不能互相干扰

# 持续性

- 持续性也称永久性（Permanence）
  - 一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。
  - 接下来的其他操作或故障不应该对其执行结果有任何影响。

# 破坏事务ACID的因素

- 多个事务并行运行时，不同事务的操作交叉执行

数据库管理系统必须保证多个事务的交叉运行不影响这些事务的隔离性

- 事务在运行过程中被强行停止

数据库管理系统必须保证被强行终止的事务对数据库和其他事务没有任何影响

# 事务执行的隔离级别

- 隔离级别高--》程序运行安全--》并发程度低
- 隔离级别的选择是每个应用系统的选择。
- 隔离级别最高的 (“serializable”) = 具有ACID 特性.

# 如何设置事务的隔离级别？

- **SET TRANSACTION ISOLATION LEVEL *X***

  **where *X* =**

  1. **SERIALIZABLE**
  2. **REPEATABLE READ**
  3. **READ COMMITTED**
  4. **READ UNCOMMITTED**

# 事务的隔离级别

- 每个事务规定的隔离级别只是针对它自己本身，与其它事务无关。
- 一个设置为可重复读的事务，它保证第一次读到的数据，以后可以多次读。该数据被封锁，不让其它用户修改它。

# 四个隔离级别产生的效果对比

| 隔离级别 | 读脏数据 | 不可重复读 | 幻影元组 |
| --- | --- | --- | --- |
| READ UNCOMMITTED | Y | Y | Y |
| READ COMMITTED | N | Y | Y |
| REPEATABLE READ | N | N | Y |
| SERIALIZABLE | N | N | N |

- 真正的隔离降低并发度
- 不同的应用系统选择其合适的隔离级别
- 需要在正确性和并发度之间做权衡

# 举例: T1 executes with READ COMMITTED
# T2 executes with READ UNCOMMITTED.

假设R（E，F）={ （B，2），（C，3）}

T1: BEGIN TRANSACTION

S1: UPDATE R SET F = (3 * F) WHERE E = B;

S2: UPDATE R SET F = (3 * F) WHERE E = C;

COMMIT;

T2: BEGIN TRANSACTION

S3: UPDATE R SET F = (2 + F) WHERE E = C;

S4: UPDATE R SET F = (2 + F) WHERE E = B;

COMMIT;

| Stage | T1 sees | T2 sees | Committed | Uncommitte |
|---|---|---|---|---|
| Initial | B=2, C=3 | B=2, C=3 | B=2, C=3 | None |
| After S1 (1 pt) | B=6, C=3 | B=6, C=3 | B=2, C=3 | B=6 |
| After S3 (1 pt) | B=6, C=3 | B=6, C=5 | B=2, C=3 | B=6, C=5 |
| After S2 (1 pt) | B=6, C=9 | B=6, C=9 | B=2, C=3 | B=6, C=9 |
| After T1 commits | | B=6, C=9 | B=6, C=9 | None |
| After S4 (2 pts) | | B=8, C=9 | B=6, C=9 | B=8, C=9 |
| After T2 commits | | | B=8, C=9 | None |

假设系统执行句子的顺序是： **S1, S3, S2, T1: COMMIT, S4, T2: COMMIT.**

# 维护ACID——事务管理器

- 恢复管理部件
  - 保证事务的原子性和永久性
- 并发控制管理部件
  - 负责事务的并发控制，维护事务的隔离性
- 使数据库从事务前的一致状态迁移到事务完成后的一致状态

# 什么是数据库的恢复技术

- 由于不同的故障（硬件、软件和人为因素造成的），使计算机非正常的中断，如何把数据库**从错误状态恢复到某一已知的正确状态**，这就是数据库的恢复技术。

# 数据库恢复概述

- 故障是不可避免的
  - 计算机硬件故障
  - 软件的错误
  - 操作员的失误
  - 恶意的破坏
- 故障的影响
  - 运行事务非正常中断，影响数据库中数据的正确性
  - 破坏数据库，全部或部分丢失数据

# 数据库恢复概述（续）

- 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)的功能，这就是数据库的恢复管理系统对故障的对策
- 恢复子系统是数据库管理系统的一个重要组成部分
- 恢复技术是衡量系统优劣的重要指标

# 故障的种类

1.事务内部的故障

2.系统故障

3.介质故障

4.计算机病毒

# 事务内部的故障

- 预期的：通过事务程序本身发现的(见下面转账事务的例子)
- 非预期的：不能由事务程序处理的。

# 事务内部的故障（续）

- 例如，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*/
    IF(BALANCE < 0 ) THEN
        {打印‘金额不足，不能转账’;
                                /*事务内部可能造成事务被回滚的情况*/
         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　　非预期的故障

- 非预期的：
  - 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而被终止等

一般，事务故障仅指这类非预期的故障

# 事务内部的故障（续）

- 事务故障意味着
  - 事务没有达到预期的终点(COMMIT或者显式的ROLLBACK)
  - 数据库可能处于不正确状态。
- 事务故障的恢复：事务撤消（UNDO）
  - 强行回滚（ROLLBACK）该事务
  - 撤销该事务已经作出的任何对数据库的修改，使得该事务象根本没有启动一样

# 系统故障

- 系统故障：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
- 数据库管理系统代码错误
- 系统断电

# 系统故障的恢复

- 一些尚未完成的事务的结果可能已送入物理数据库，造成数据库可能处于不正确状态。

  - 恢复策略：系统重新启动时，恢复程序让所有非正常终止的事务回滚，强行撤消（UNDO）所有未完成事务

# 系统故障的恢复

- 有些已完成的事务可能有一部分甚至全部留在缓冲区，尚未写回到磁盘上的物理数据库中，系统故障使得这些事务对数据库的修改部分或全部丢失

  - 恢复策略：系统重新启动时，恢复程序需要重做（REDO）所有已提交的事务

# 介质故障

- 介质故障： 硬故障，指外存故障
  - 磁盘损坏
  - 磁头碰撞
  - 瞬时强磁场干扰
- 介质故障破坏数据库或部分数据库，并影响正在存取这部分数据的所有事务
- 介质故障比前两类故障的可能性小得多，但破坏性大得多

# 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，造成对计算机系统包括数据库的危害
- 计算机病毒种类
  - 小的病毒只有20条指令，不到50B
  - 大的病毒像一个操作系统，由上万条指令组成

# 计算机病毒（续）

- 计算机病毒的危害
  - 有的病毒传播很快，一旦侵入系统就马上摧毁系统
  - 有的病毒有较长的潜伏期，计算机在感染后数天或数月才开始发病
  - 有的病毒感染系统所有的程序和数据
  - 有的只对某些特定的程序和数据感兴趣
- 计算机病毒已成为计算机系统的主要威胁，自然也是数据库系统的主要威胁
- 数据库一旦被破坏仍要用恢复技术把数据库加以恢复

# 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

- 恢复操作的基本原理：冗余
  - 利用存储在系统别处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
- 恢复的实现技术：复杂
  - 一个大型数据库产品，恢复子系统的代码要占全部代码的10%以上

# 恢复的实现技术

## 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）
    - 登记日志文件（logging）
2. 如何利用这些冗余数据实施数据库恢复

# 恢复的实现技术：数据转储

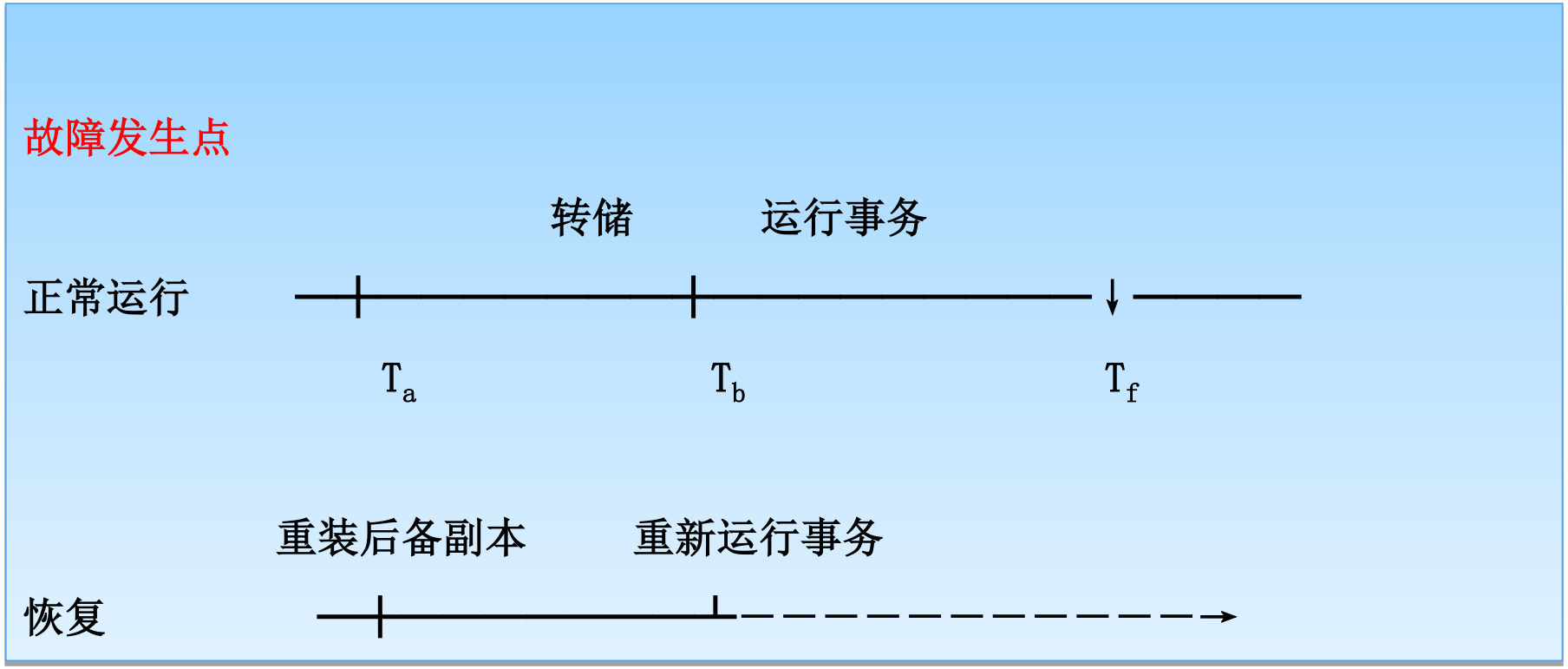
- 转储：数据库管理员定期地将整个数据库复制到磁带、磁盘或其他存储介质上保存起来的过程

- 备用的数据文本称为后备副本(backup)或后援副本

# 数据转储（续）

- 重装后备副本只能将数据库恢复到转储时的状态
- 要想恢复到故障发生时的状态，必须重新运行自转储以后的所有更新事务

# 数据转储实例




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，首先由数据库管理员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

# 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

# （1）静态转储与动态转储

- 静态转储
  - 无运行事务时进行的转储操作
  - 转储期间不允许对数据库的任何存取、修改活动
  - 转储开始与结束：数据处于一致性状态
  - 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

# 静态转储与动态转储（续）

- 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。
      后备副本上的A过时了

# 静态转储与动态转储（续）

- 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就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
  - 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

| 转储方式 | 转储状态 | |
|---|---|---|
| | 动态转储 | 静态转储 |
| 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 |

# 恢复的实现技术：登记日志文件

1.日志文件的格式和内容

2.日志文件的作用

3.登记日志文件

# 1.日志文件的格式和内容

- 什么是日志文件
  - 日志文件(log file)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）

- 日志记录包含:
  - 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ID、Block NO.）
  - 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
  - 被更新的数据块

# 例如

| (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> |

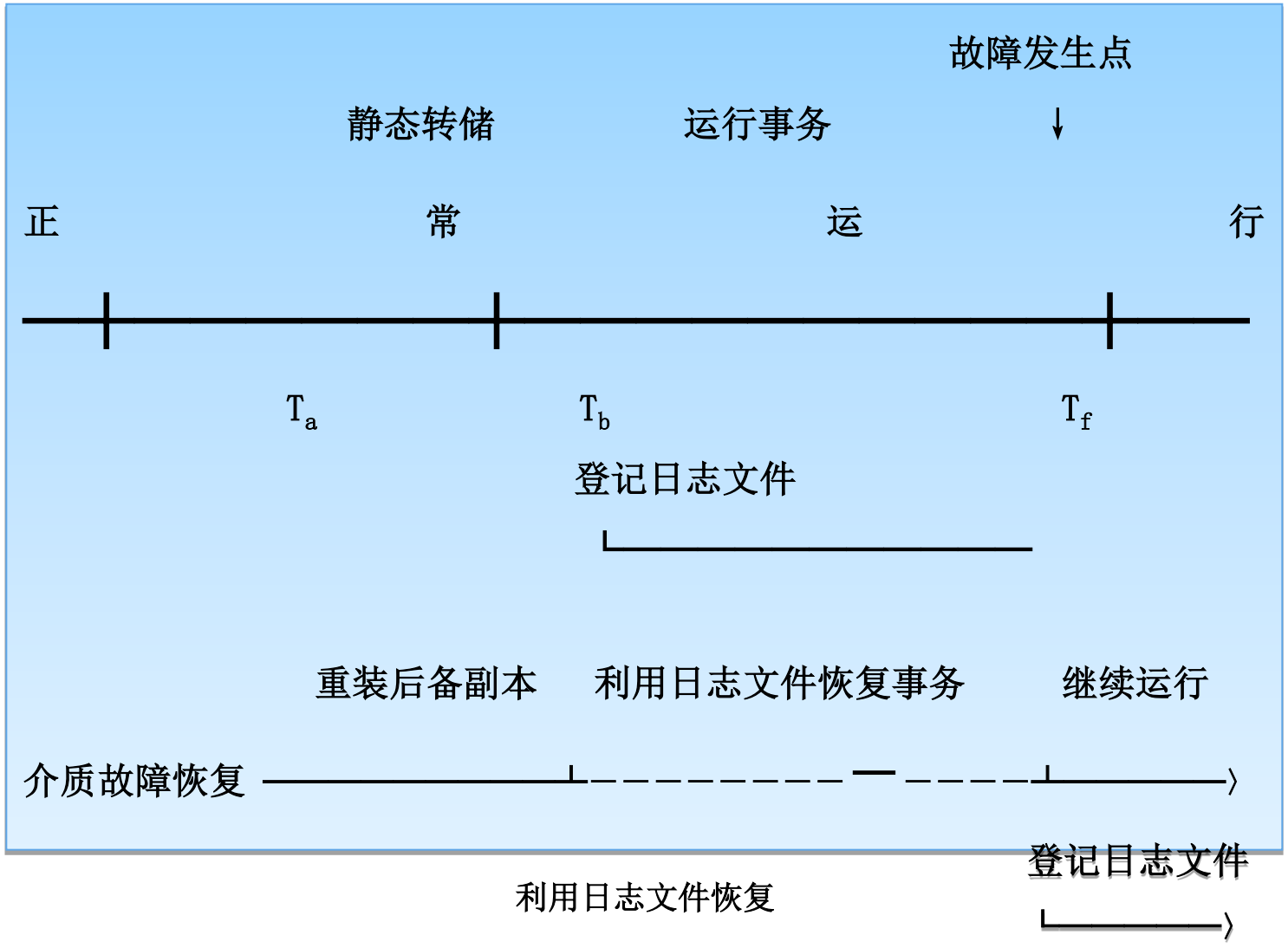
# 2.日志文件的作用

- 用途
  - 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

# 日志文件的作用（续）

- 具体作用
  - 事务故障恢复和系统故障恢复必须用日志文件。
  - 在动态转储方式中必须建立日志文件，后备副本和日志文件结合起来才能有效地恢复数据库。

# 日志文件的作用（续）


故障发生点
静态转储
运行事务
正
常
运
行
$T_a$
$T_b$
$T_f$
登记日志文件
重装后备副本
利用日志文件恢复事务
继续运行
介质故障恢复
利用日志文件恢复
登记日志文件

# 3.登记日志文件

- 为保证数据库是可恢复的，登记日志文件时必须遵循两条原则
  - 登记的次序严格按并发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中
    - 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

# 恢复策略

- 事务故障的恢复
- 系统故障的恢复
- 介质故障的恢复

# 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（续）

（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

- 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（续）

（2） 对撤销(UNDO)队列事务进行撤销(UNDO)处理

- 反向扫描日志文件，对每个撤销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，对每个重做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数据库管理员介入

- 数据库管理员的工作
  - 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数据库管理系统完成

# 具有检查点的恢复技术

1.问题的提出

2.检查点技术

3.利用检查点的恢复策略

# 问题的提出

- 两个问题
  - 搜索整个日志将耗费大量的时间
  - 重做处理：重新执行，浪费了大量时间

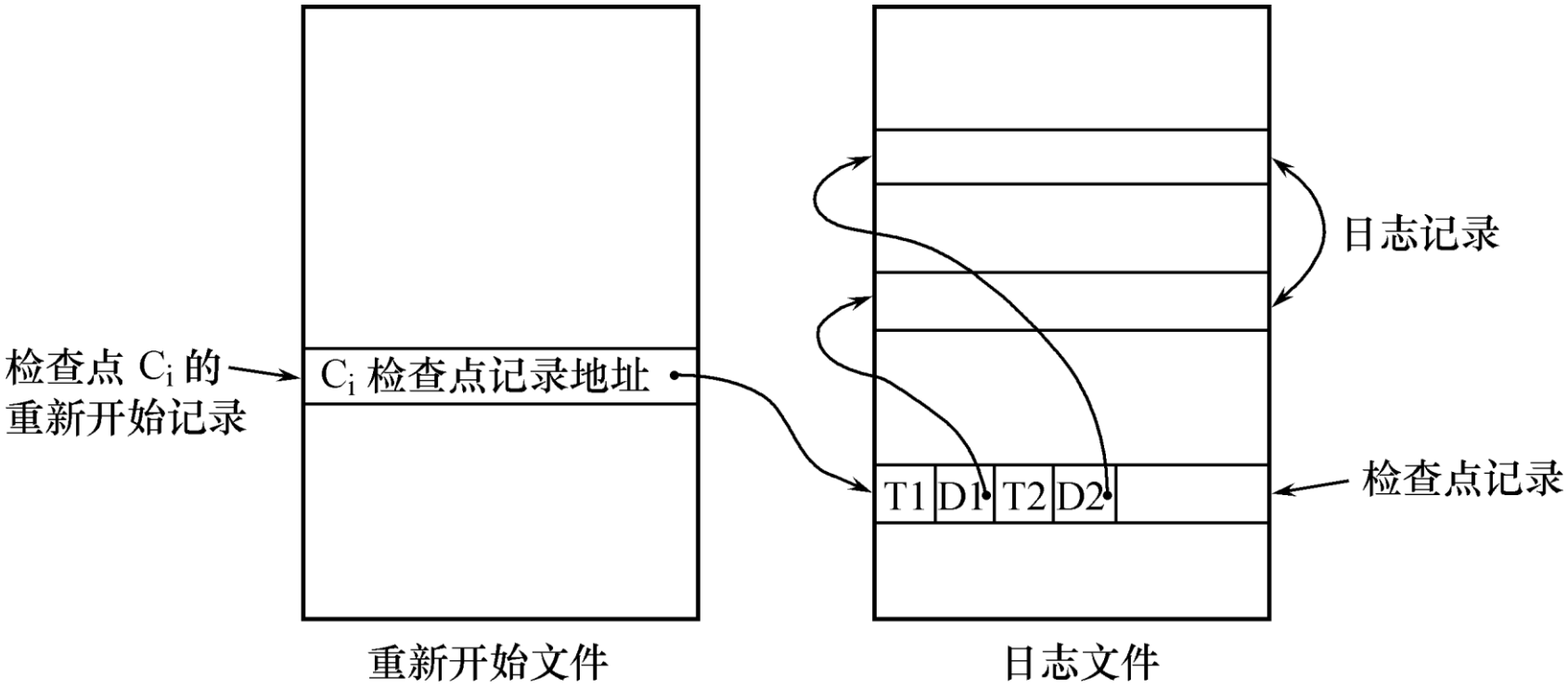
# 解决方案

- 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

# 检查点技术

- 检查点记录的内容
  - 建立检查点时刻所有正在执行的事务清单
  - 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

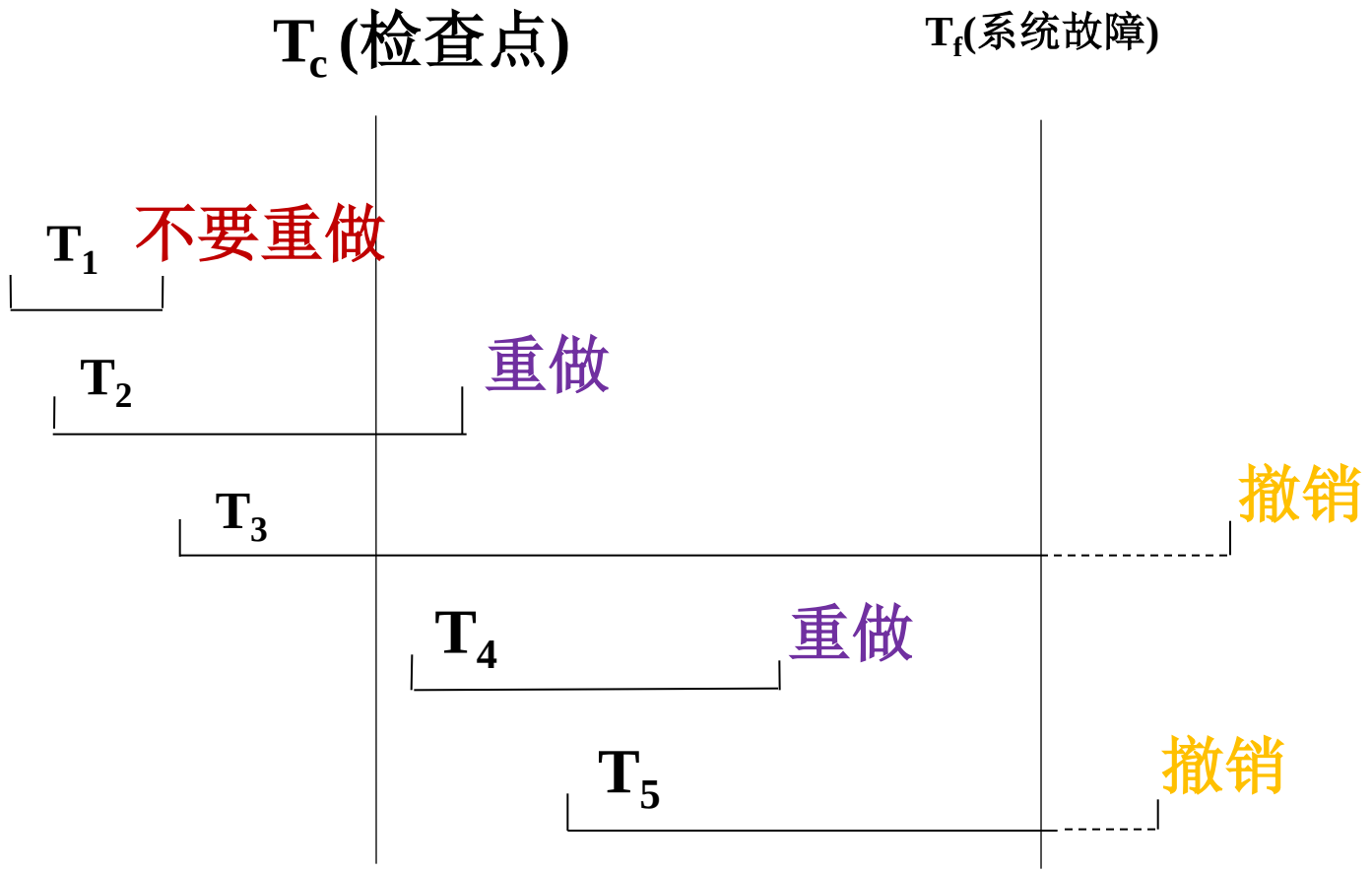
  （4）把检查点记录在日志文件中的地址写入一个重新开始文件

# 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

# 利用检查点的恢复策略

**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;直到日志文件结束

（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

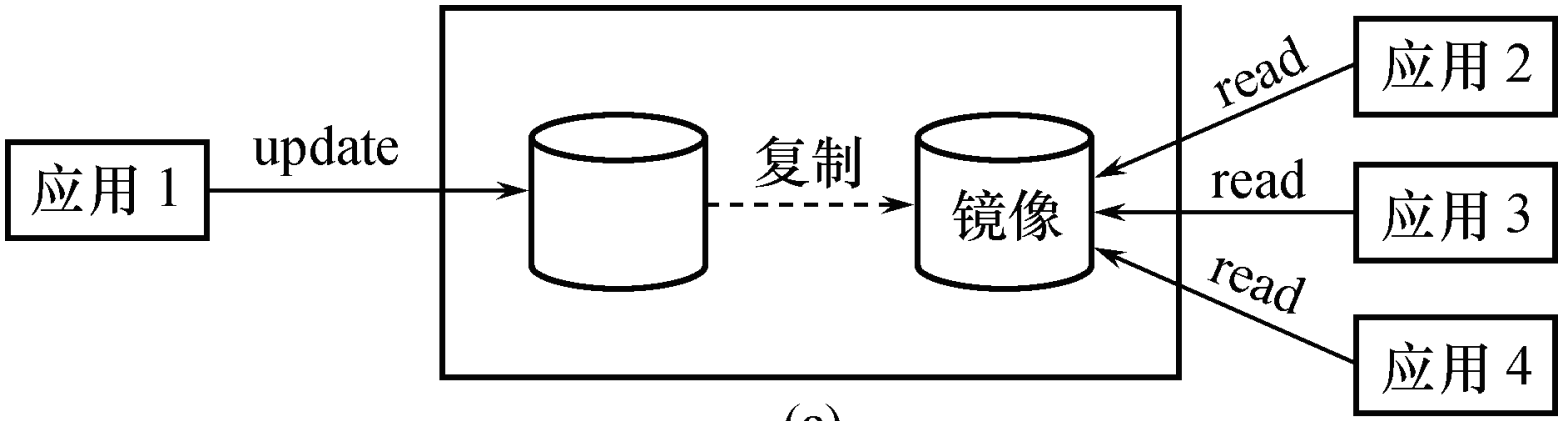
# 数据库镜像

- 介质故障严重影响数据库的可用性
  - 恢复费时
  - 数据库管理员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
  - 数据库管理系统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
  - 数据库管理系统自动保证镜像数据与主数据的一致性

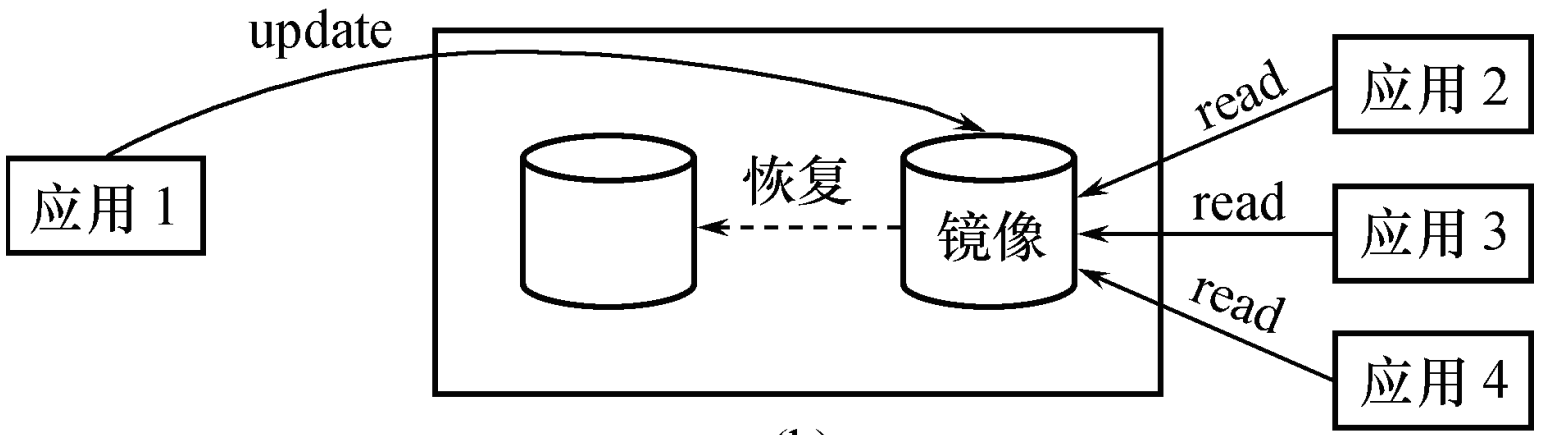
    每当主数据库更新时，数据库管理系统自动把更新后的数据复制过去




(a)

# 数据库镜像的用途

- 介质故障
  - 可由镜像磁盘继续提供使用
  - 同时数据库管理系统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



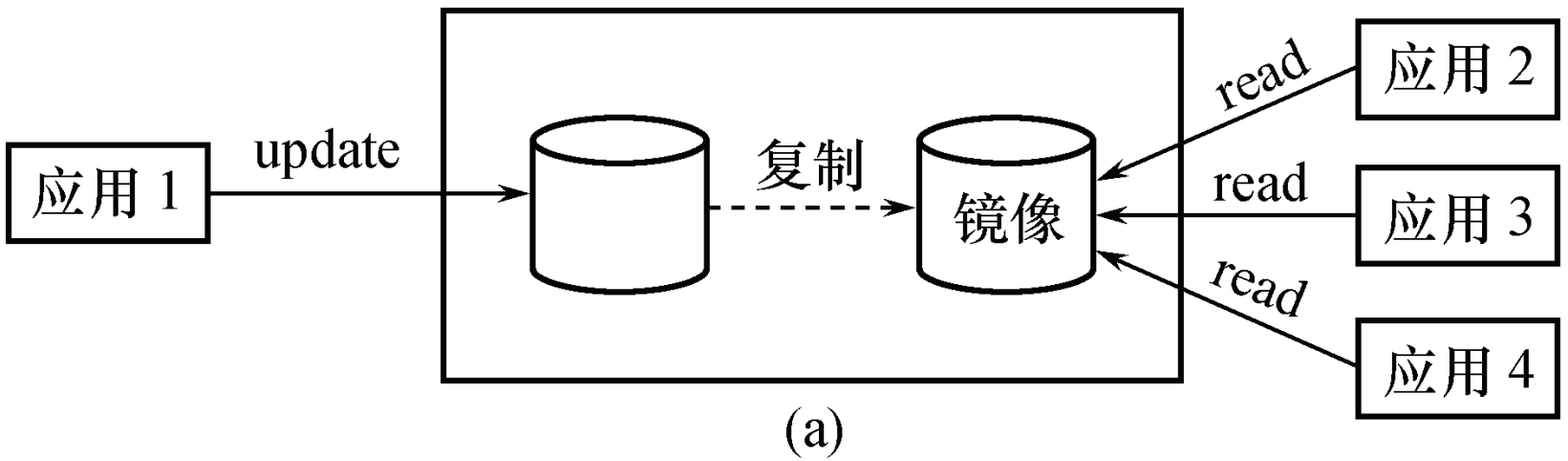

(b)

# 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




(a)

# 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
  - 不是对整个数据库进行镜像

# 总结：数据库的恢复技术

- 事务的概念和特点
- 故障的种类
- 数据库的恢复

① 日志文件

② 检查点恢复方法

- 数据转贮和建立镜像