

# 数据库原理

## The Theory of Database System

### 第六章 数据库保护



中国矿业大学计算机学院



中国矿业大学数据库原理精品课程

- 数据库系统中的数据是由**DBMS**统一管理和控制的，为了适应数据共享的环境，**DBMS**必须提供数据保护能力，以保证数据库中数据的安全可靠和正确有效。
- 数据保护
  - 安全性
  - 完整性
  - 并发控制
  - 数据库恢复



# 第六章 数据库保护

- 6.1 事务
- 6.2 数据库恢复
- 6.3 并发控制
- 6.4 数据库安全性
- 6.5 数据库完整性



# 6.1 事务

- 一、什么是事务
- 二、如何定义事务
- 三、事务的特性



# 一、什么是事务

- 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
  - 一个应用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



## 二、如何定义事务

- 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



# 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永久生效

## 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  
回滚事务的所有更新操作
- 事务回滚到开始时的状态



```
BEGIN TRANSACTION
```

```
  read(Balance); //读账户甲的余额Balance;
```

```
  Balance=Balance-Amount
```

```
  if( Balance<0) Then
```

```
    { print “金额不足，不能转账”;
```

```
      Rollback; }
```

```
  else
```

```
    { read(Balance); //读账户乙的余额Balance;
```

```
      Balance=Balance+Amount;
```

```
      write(Balance);
```

```
      commit; }
```





# SQL语言分类

- **DDL(数据定义语言)**
  - Create, Alter, Drop 等语句自动提交, 无需用Commit提交。
- **DQL(数据查询语言)**
  - Select 查询语句不存在提交问题。
- **DML(数据操纵语言)**
  - Insert、Update、Delete 这些语句需要Commit才能提交。
- **DTL(事务控制语言)**
  - Commit、Rollback 事务提交与回滚语句。
- **DCL(数据控制语言)**
  - Grant、Revoke 授予权限与回收权限语句。



# 三、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）



# 1. 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
- 保证原子性由**DBMS**的事务子系统来实现



## 2. 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：**

数据库中包含失败事务的结果



# 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

A	B
$A=A-1$	$B=B+1$

- 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，数据库就处于不一致性状态。

一致性一般由包含事务的应用程序来完成的，也可以由系统检查完整性约束来自动完成。



### 3. 隔离性

对并发执行而言

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



$T_1$	$T_2$
① 读A=16	读A=16
②	
③ $A \leftarrow A-1$ 写回A=15	
④	$A \leftarrow A-3$ 写回A=13

隔离性是由DBMS的并发控制子系统实现的。



## 4. 持续性

- 持续性也称持久性（**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。
- 事务的持久性是由**DBMS**的恢复管理子系统实现。





# 事务的特性

- 保证事务**ACID**特性是事务处理的任务
- 破坏事务**ACID**特性的因素
  - 多个事务并行运行时，不同事务的操作交叉执行
  - 事务在运行过程中被强行停止



# 第六章 数据库保护

6.1 事务

6.2 数据库恢复

6.3 并发控制

6.4 数据库安全性

6.5 数据库完整性



## 6.2 数据库恢复

- 故障是不可避免的
  - 计算机硬件故障
  - 系统软件和应用软件的错误
  - 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
  - 破坏数据库



# 数据库恢复（续）

- 数据库管理系统对故障的对策
  - 保证事务**ACID**
  - **DBMS**提供恢复子系统
  - 保证故障发生后，能把数据库中的数据从错误状态恢复到某种逻辑一致的状态
- 恢复技术是衡量系统优劣的重要指标



## 6.2.1 故障的种类

- 事务故障
  - 系统故障
  - 介质故障
  - 计算机病毒
- } 软故障
- 硬故障



# 一、事务故障

- 什么是事务故障
  - 某个事务在运行过程中由于种种原因未运行至正常结束点就终止了
- 事务故障的常见原因
  - 输入数据有误
  - 运算溢出
  - 违反了某些完整性限制
  - 某些应用程序出错
  - 并行事务发生死锁
  - .....



## 二、系统故障

- 什么是系统故障  
造成系统停止运转并要求系统重新启动的事件。
  - 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
  - 外部存储设备上的数据未受影响



# 系统故障的常见原因

- 操作系统或DBMS代码错误
- 操作员操作失误
- 特定类型的硬件错误（如CPU故障）
- 突然停电





### 三、介质故障

- 硬件故障使存储在外存中的数据部分丢失或全部丢失
- 介质故障比前两类故障的可能性小得多，但破坏性大得多



# 介质故障的常见原因

- 硬件故障
  - 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



## 四、计算机病毒

- 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。



## 6.2.2 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



# 恢复机制涉及的关键问题

1. 如何建立冗余数据
  - 数据转储（backup）
  - 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

一、什么是转储

二、转储的用途

三、转储方法

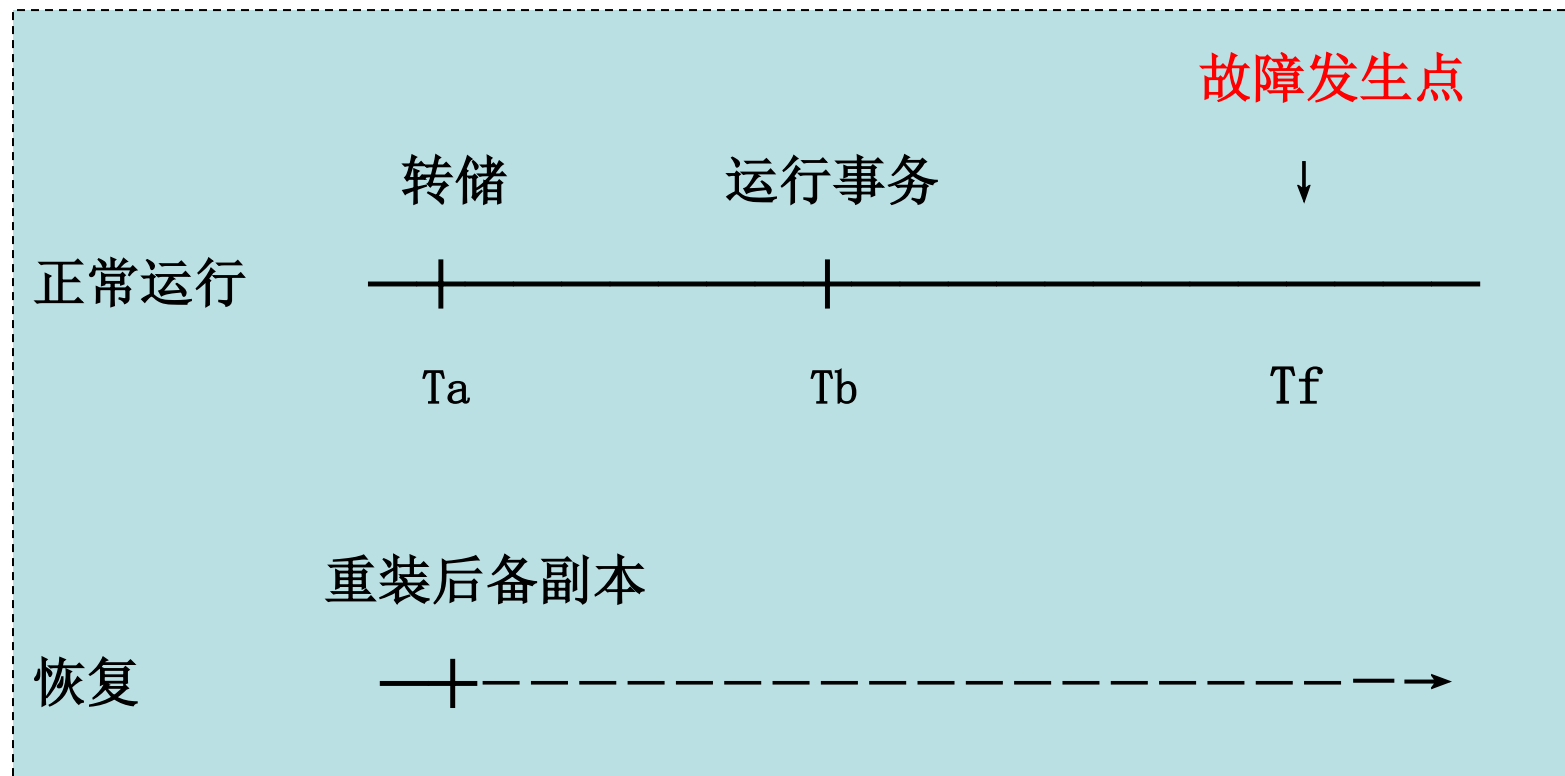


# 一、什么是转储

- 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。
- 这些备用的数据文本称为后备副本或后援副本。



# 转储





# 三、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

- 在系统中无运行事务时进行转储
- 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
- 缺点：降低了数据库的可用性
  - 转储必须等用户事务结束
  - 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

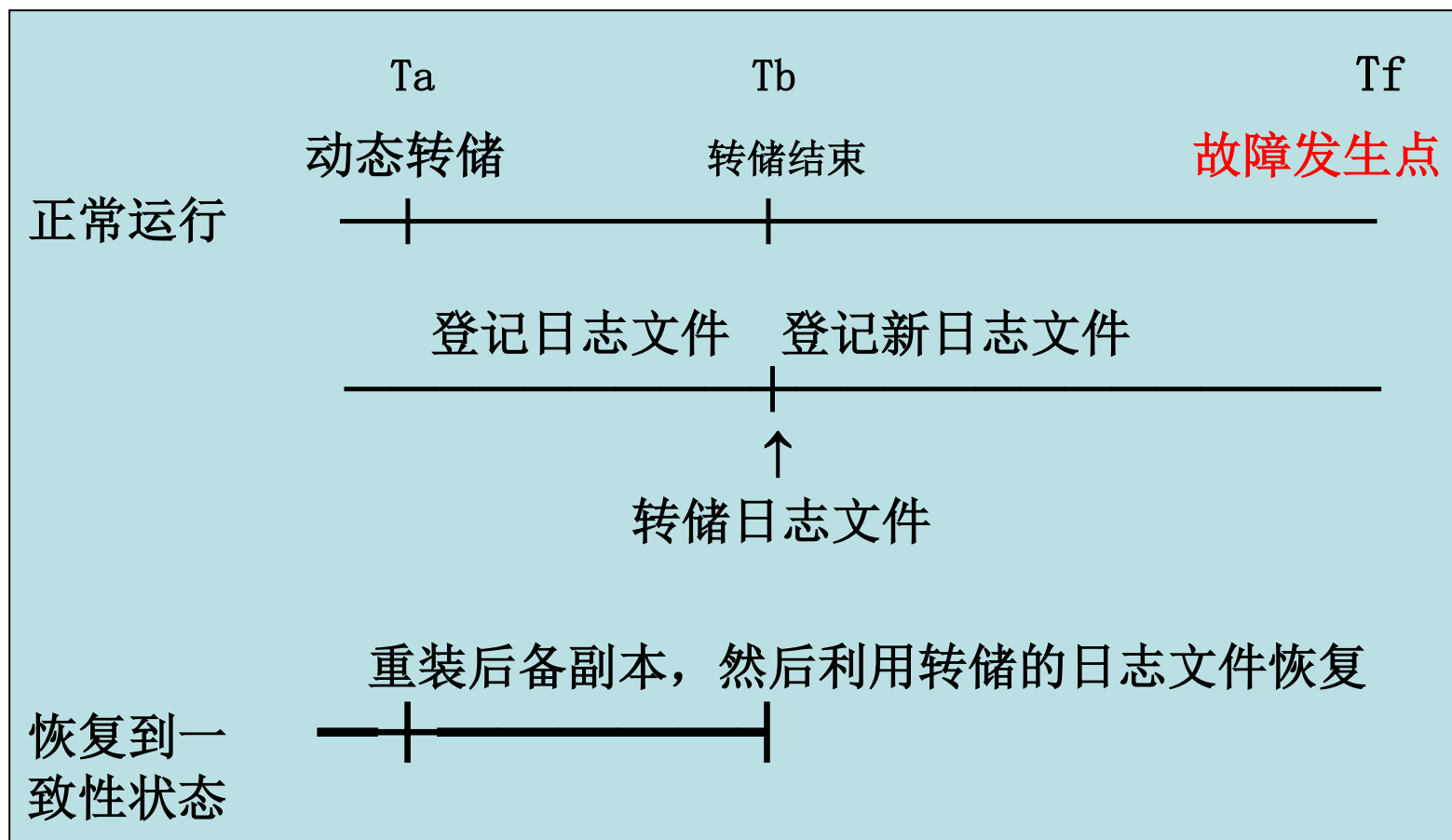


# 动态转储

- 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



# 利用动态转储副本进行恢复



## 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



# 转储策略

- 应定期进行数据转储，制作后备副本。
- 转储十分耗费时间和资源的，不能频繁进行。
- **DBA**应该根据数据库使用情况确定适当的转储周期和转储方法。

例：

- 每天晚上进行动态增量转储
- 每周进行一次动态海量转储
- 每月进行一次静态海量转储





# 日志文件

- 一、日志文件的内容
- 二、日志文件的用途
- 三、登记日志文件的原则



# 一、日志文件的内容

## 1. 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

## 2. 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



# 日志文件的内容（续）

## 3. 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
- 与事务有关的内部更新操作



## 4. 日志记录 (log record)

### 每条日志记录的内容

- 日志编号 (**LSN**)
- 事务标识
- 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 (**undo**)
  - 对插入操作而言, 此项为空值
- 更新后数据的新值 (**redo**)
  - 对删除操作而言, 此项为空值

基于记录

基于数据块



# 日志文件的格式

事务标识	操作类型	对象标识	前像	后像
------	------	------	----	----

事务T开始，日志记录为（T，START...）

事务T修改对象A，日志记录为（T, UPDATE, A, 前像, 后像）

事务T插入对象A，日志记录为（T，INSERT，A，，后像）

事务T删除对象A，日志记录为（T，DELETE，A，前像，）

事务T提交，日志记录为（T，COMMIT...）

事务T回滚，日志记录为（T，ROLLBACK...）

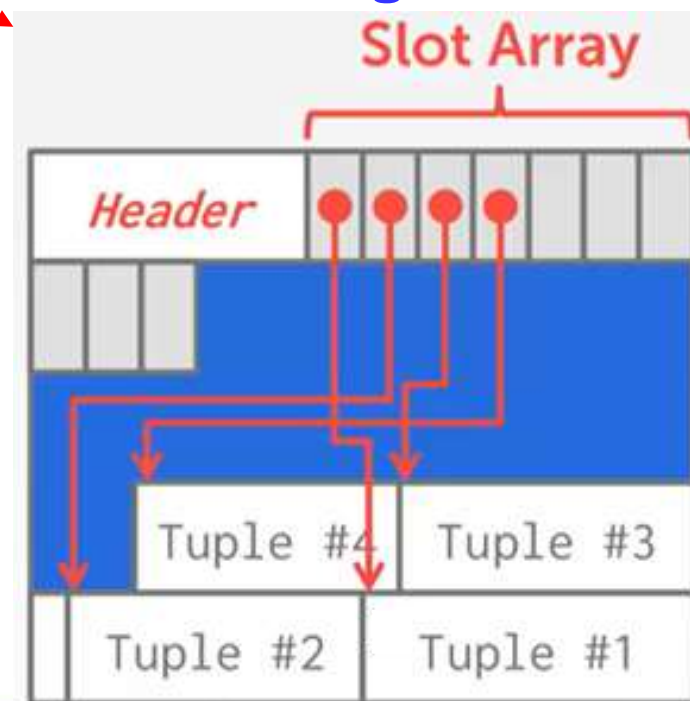


LSN	prevLSN	TxnId	Type	Object	Before	After	UndoNext
001	nil	$T_1$	BEGIN	-	-	-	-
002	001	$T_1$	UPDATE	A	30	40	-
⋮							
011	002	$T_1$	ABORT	-	-	-	-
⋮							

说明:

- 数据库中的数据简化记为A, B.....
- 将事务中的操作简化记为R(A), W(A).....

Page



**T1**  
BEGIN  
W(A)  
W(B)  
COMMIT

**BUFFER**

**DISK**

**LOG**

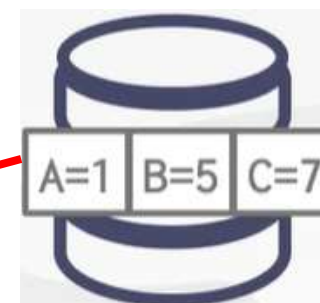
<T1, BEGIN>  
<T1, A, 1, 8>  
<T1, B, 5, 9>  
<T1, COMMIT>

<T1, BEGIN>  
<T1, A, 1, 8>  
<T1, B, 5, 9>  
<T1, COMMIT>

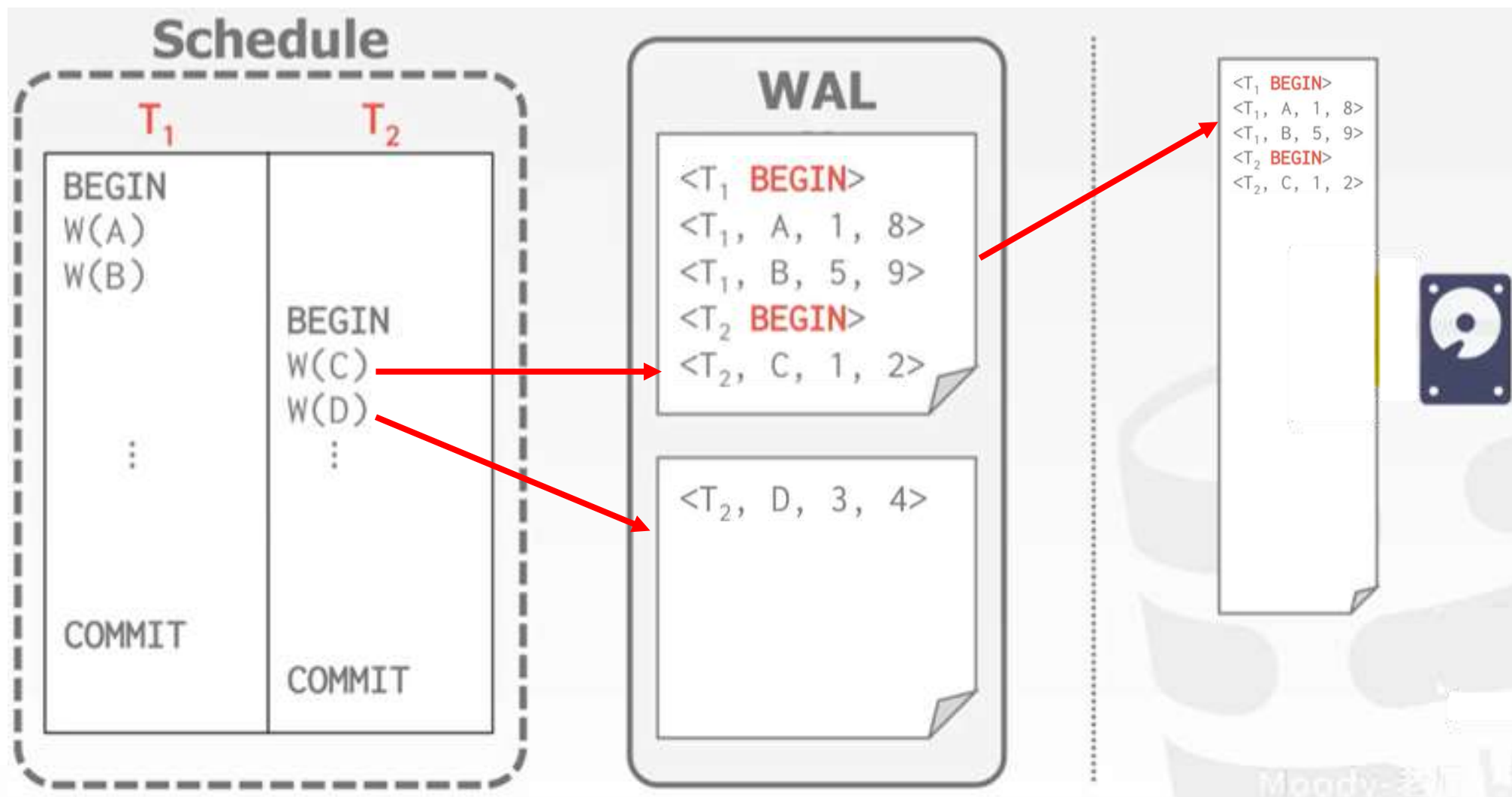
**WAL:**  
**write-ahead log**

**DATA**

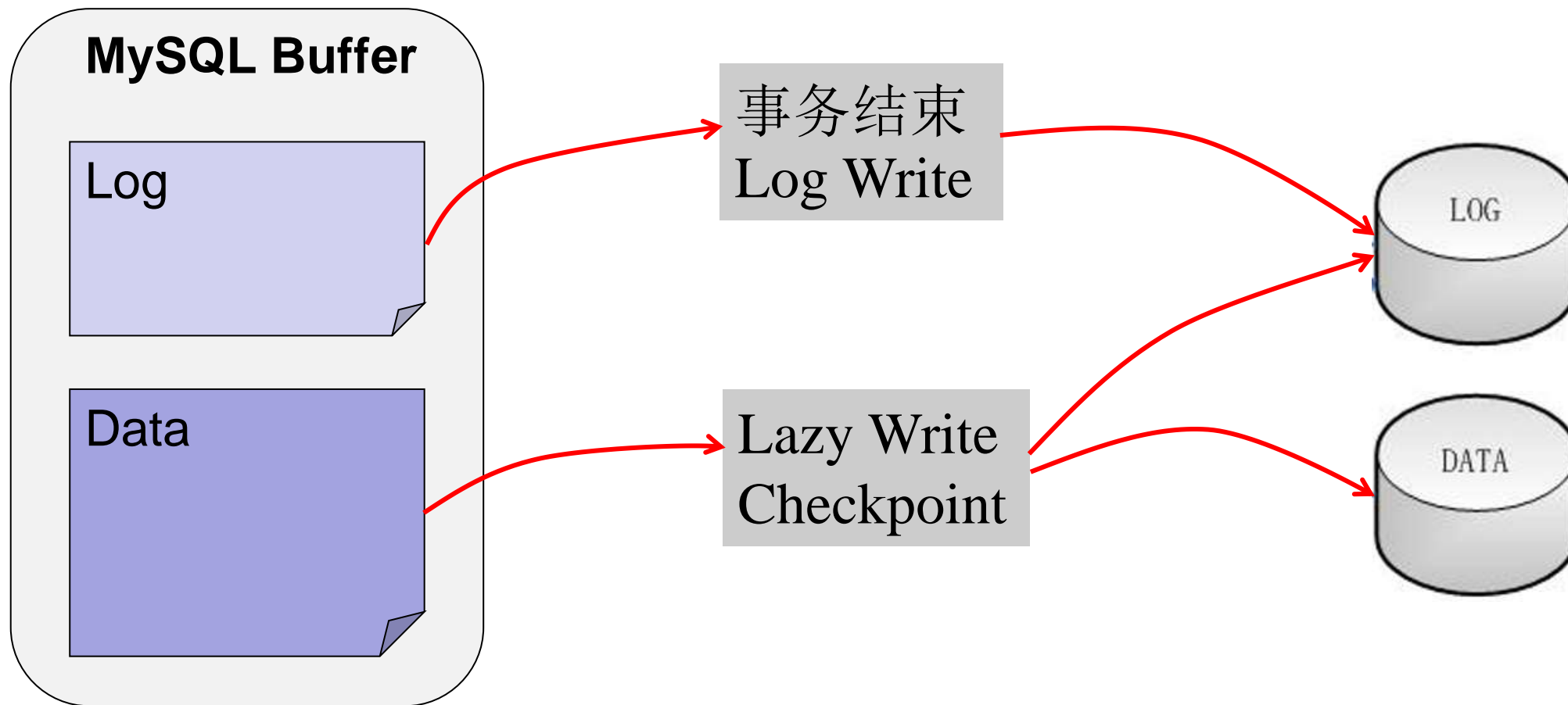
**A=8** **B=9** **C=7**











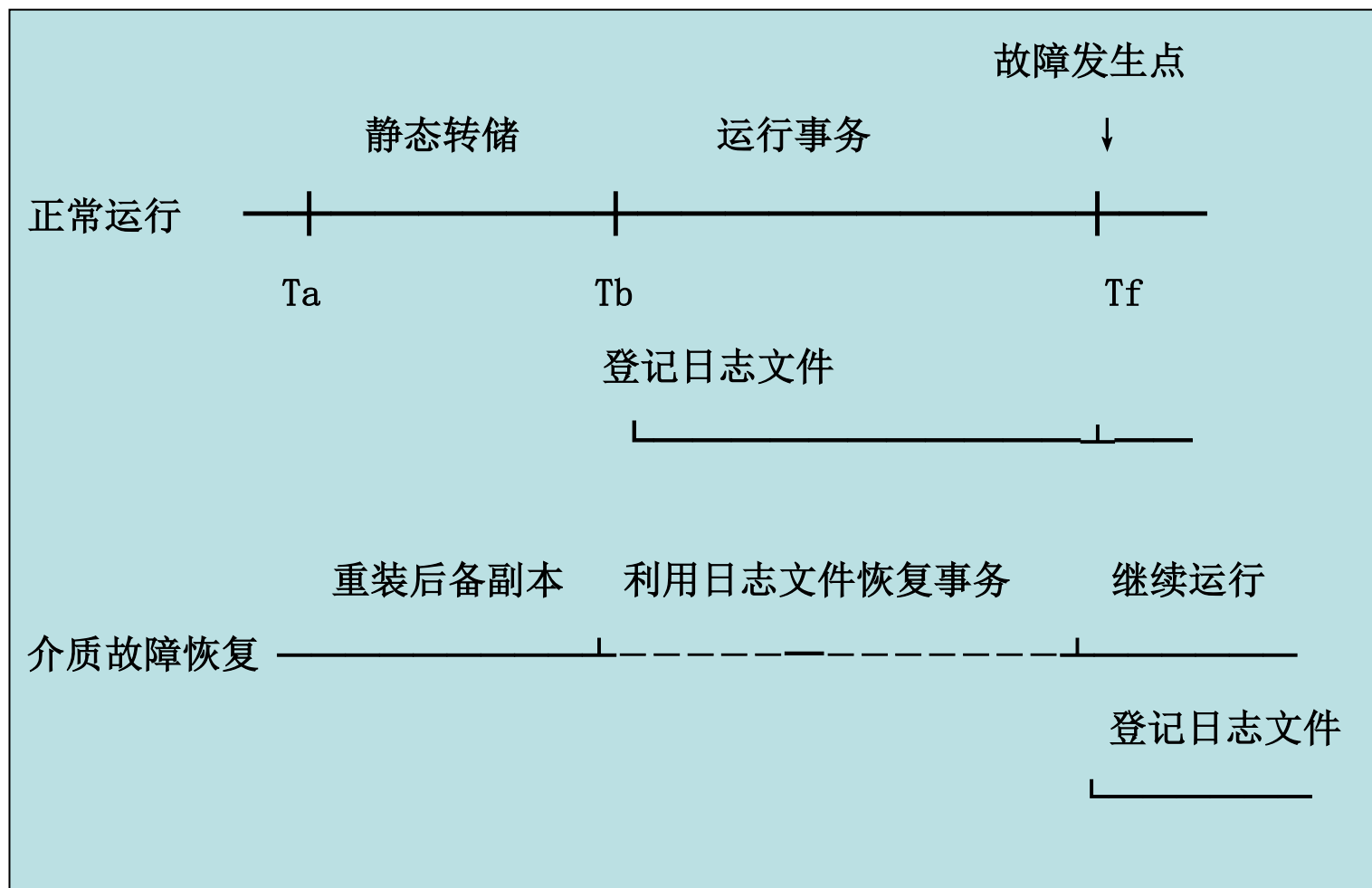
## 二、日志文件的用途

### 1. 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



# 日志文件的用途（续）



# 日志文件的处理

## 1、undo操作

- 对尚未结束的事务进行撤消
- 方法：旧值替代新值
- 反向扫描日志文件，按照登记的顺序反向撤销每笔操作。



## 2、redo操作

- 对已经提交的事务进行重做
- 方法：新值替代旧值
- 正向扫描日志文件，根据日志文件中的操作次序，重新执行每项操作。



# 三、登记日志文件的原則

- 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行事务执行的时间次序
  - 必须先写日志文件，后写数据库（**WAL**）
    - 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



# 恢复策略

- 1、事务故障的恢复
- 2、系统故障的恢复
- 3、介质故障的恢复



# 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中止
- 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





# 事务故障的恢复策略

确定事务编号后，反向扫描日志文件，将日志中更新前的数据（**Befor Image, BI**）写回到数据库中，直至事务的开始标志。

- 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，则用**BI** 代替 **AI**（**After Image**）



## 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
  - 一些未完成事务对数据库的更新已写入数据库
  - 一些已提交事务对数据库的更新还留在缓冲区还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



# 系统故障的恢复策略

1. 正向扫描日志文件（即从头扫描日志文件）
  - Redo队列: 在故障发生前已经提交的事务  
T1, T3, T8.....
  - Undo队列: 故障发生时尚未完成的事务  
T2, T4, T5, T6, T7, T9 .....



# 系统故障的恢复策略

## 2. 对Undo队列事务进行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。

## 3. 对Redo队列事务进行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。



### 3 介质故障的恢复

1. 重装数据库，使数据库恢复到一致性状态
2. 利用日志文件重做已完成的事务

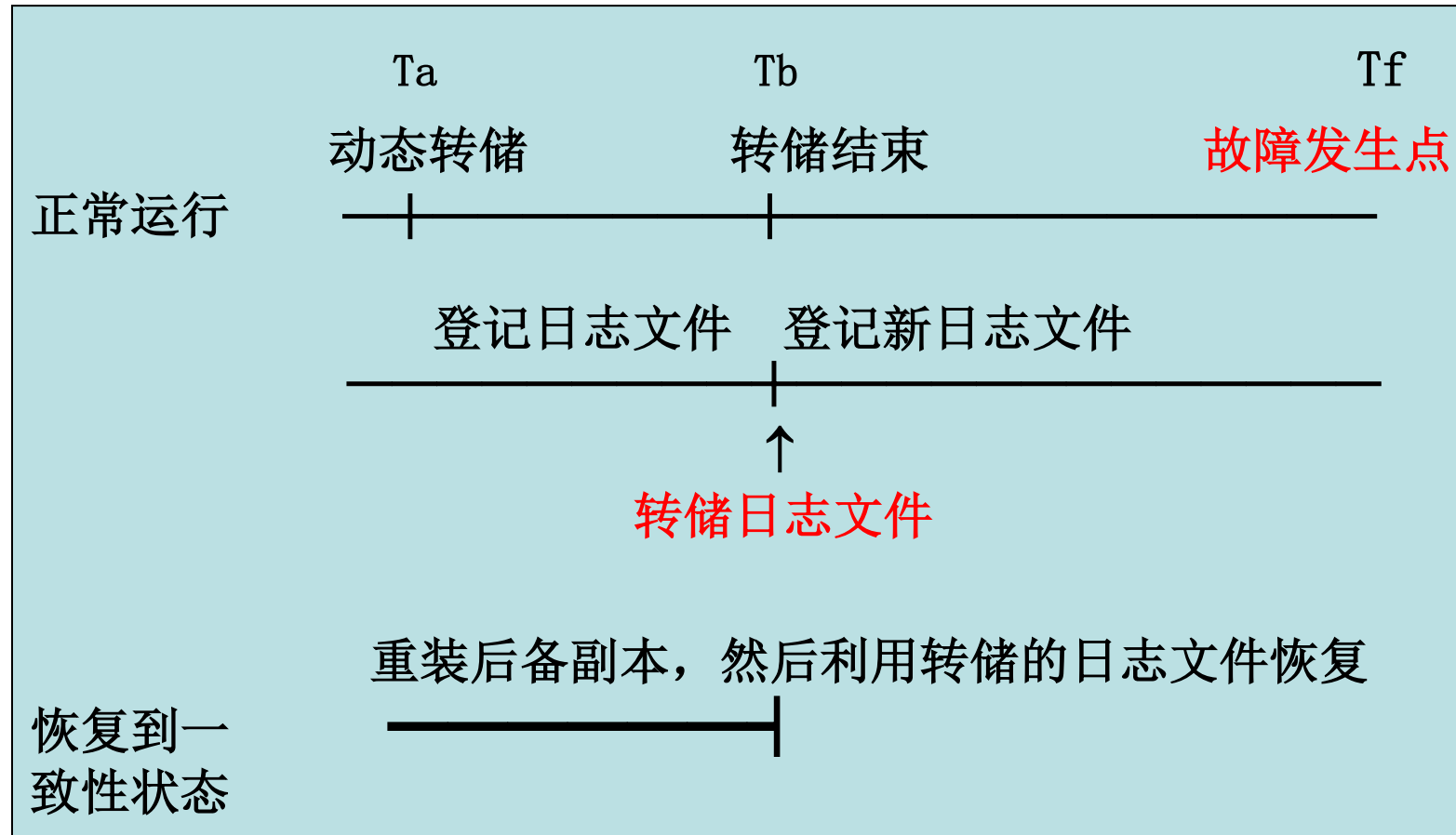


# 介质故障的恢复策略

1. 装入最新的后备数据库副本，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相同的方法（即 REDO+UNDO），才能将数据库恢复到一致性状态。



# 利用动态转储副本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，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



# 具有检查点的恢复技术

- 一、问题的提出
- 二、检查点技术
- 三、利用检查点的恢复策略



# 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

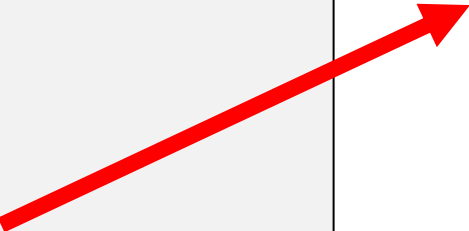


# 思考：

- 日志文件中加入检查点时需要记录什么？
- 如何找到离故障发生时最近的检查点？



```
.....  
<T1 BEGIN>  
<T1, A, 1, 2>  
<T2 BEGIN>  
<T2, A, 2, 3>  
<T1 COMMIT>  
<T3 BEGIN>  
<T3, INSERT, C, ,5>  
<T4 BEGIN>  
<T4, C, 11, 2>  
<T4 COMMIT>  
.....  
<CHECKPOINT>  
<T2, B, 9, 6>  
<T2 COMMIT>  
<T3, C, 5, 4>  
...  
...  
CRASH!
```

- 
- (1) 执行checkpoint时正在进行着的事务编号;
  - (2) 每个正在执行的事务在checkpoint之前且距离checkpoint最近一笔操作的地址;



# 解决方案

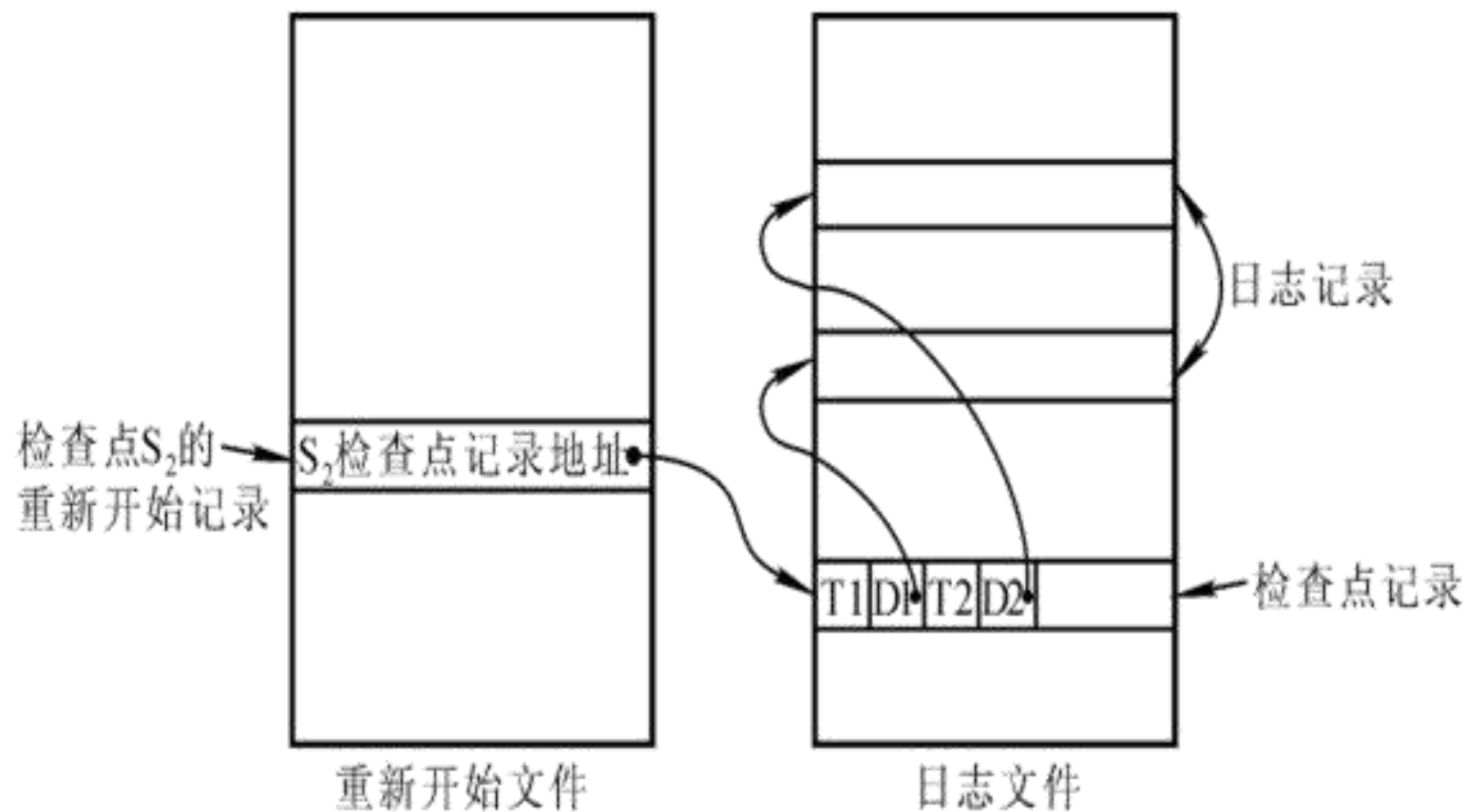
- 具有检查点（**checkpoint**）的恢复技术
  - 在日志文件中增加检查点记录（**checkpoint**）
  - 增加重新开始文件
  - 恢复子系统在登录日志文件期间动态地维护日志



## 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







# 在检查点维护日志文件

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

- 定期
  - 按照预定的一个时间间隔
- 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

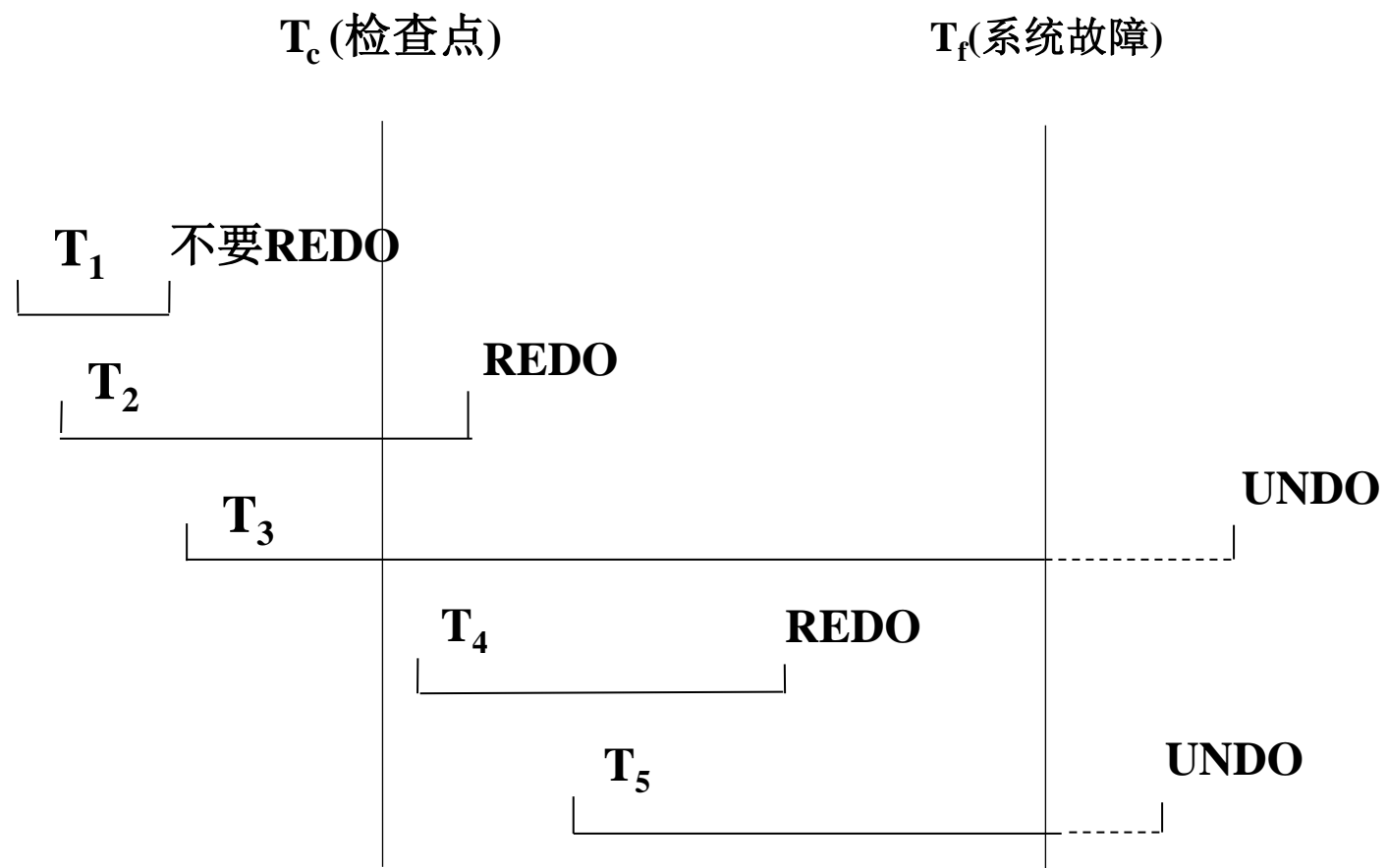


### 三、利用检查点的恢复策略

- 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
- 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）



# 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
- 2 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

- 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**）

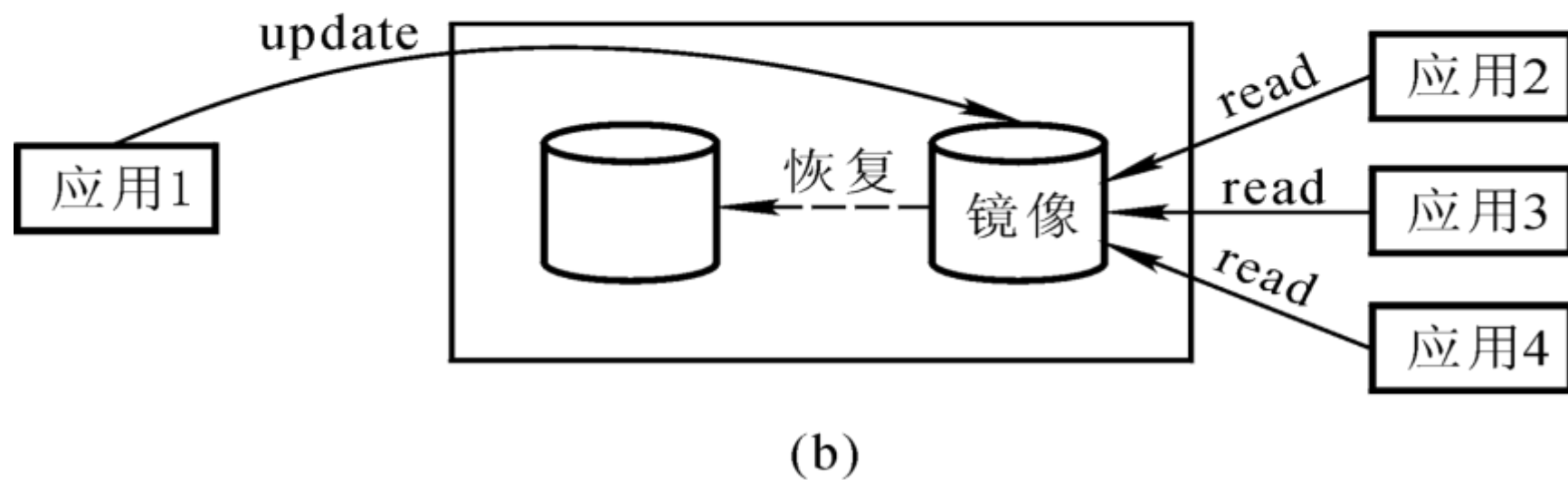
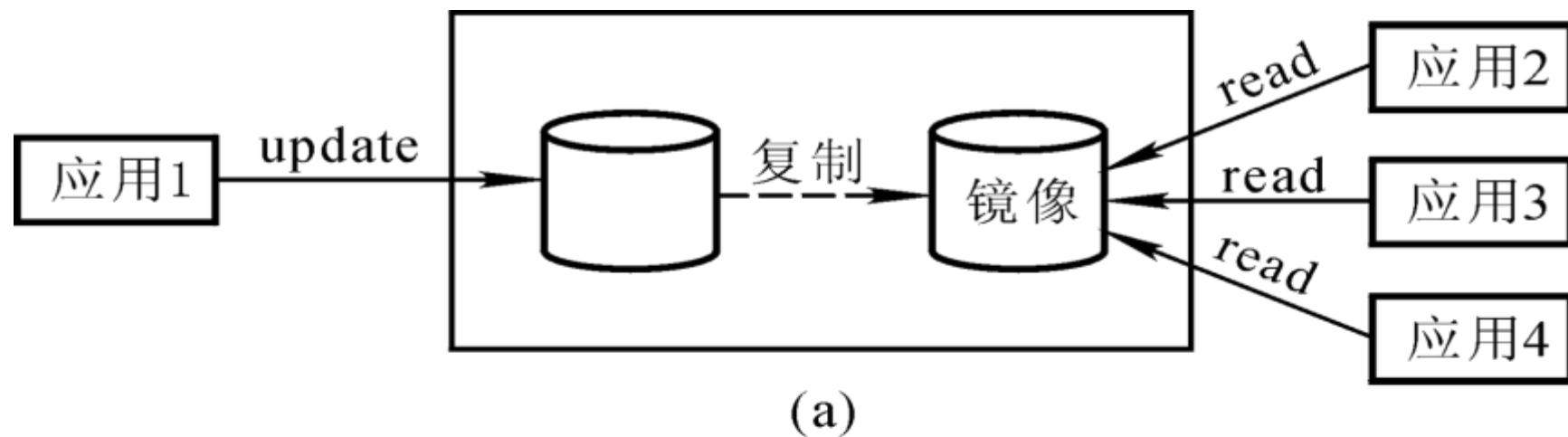




# 数据库镜像（续）

- 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的一致性





# 数据库镜像的用途

- 出现介质故障时
  - 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
- 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



# 本节重点

- 事务的概念和特性
- 数据转储的种类
- 日志文件的处理、作用以及登记原则
- 故障的种类以及对应的恢复策略
- 具有检查点的恢复技术

