



# 数据库系统概论

## An Introduction to Database System



## 第十章 数据库恢复技术

# 第十章 数据库恢复技术

**10.1 事务的基本概念**

**10.2 数据库恢复概述**

**10.3 故障的种类**

**10.4 恢复的实现技术**

**10.5 恢复策略**

**10.6 具有检查点的恢复技术**

**10.7 数据库镜像**

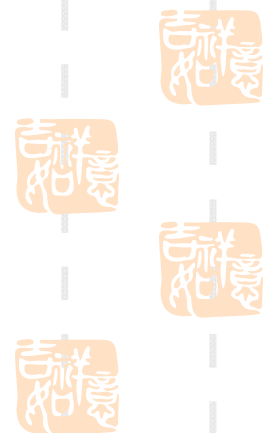
**10.8 小结**

# 10.1 事务的基本概念



## 一、事务定义

## 二、事务的特性



# 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



## 事务 (续)



可以仅仅由一个或多个操作构成  
例如交互命令状态下只有一个操作

也可以把操作序列分散在应用程序中  
在begin-trans和commit之间的操作构成  
**BEGIN TRANSACTION**

. . .

**COMMIT 或 ROLLBACK**



# 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



# 事务的提交与回滚



## ■ 提交( Commit )

- 👉 通知事务管理器一个逻辑工作单元已完成，所做的更新操作可以被提交或永久保留
- 👉 表明事务成功地结束
- 👉 已经执行有效性检验



## ■ 回滚( RollBack )



通知事务管理器事务未能正常完成，数据库可能处于不一致状态，当前事务所做的所有更新操作必须撤消

表明事务不成功地结束

# 事务的例子 银行转账

款项从 A 帐户转帐至 B 帐户

```
begin_transaction()
```

```
  read(A)
```

```
  A:=A-50
```

```
  write(A)
```

```
  If A<0
```

```
    Then
```

```
      begin
```

```
        display“ A 帐户余款不足”
```

```
        ROLLBACK
```

```
      end
```

```
    ELSE
```

```
      begin
```

```
        read(B)
```

```
        B:=B+50
```

```
        write(B)
```

```
        COMMIT
```

```
      end
```

```
end_transaction()
```



## 二、事务的特性(ACID特性)

事务的ACID特性:

- 原子性(Atomicity):

- 事务在执行时, 应“要么不做, 要么全做”, 不允许部分完成
- 事务是不可拆分的
- 事务必须以 **Commit/Rollback** 结束

- 一致性(Consistency):

- 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

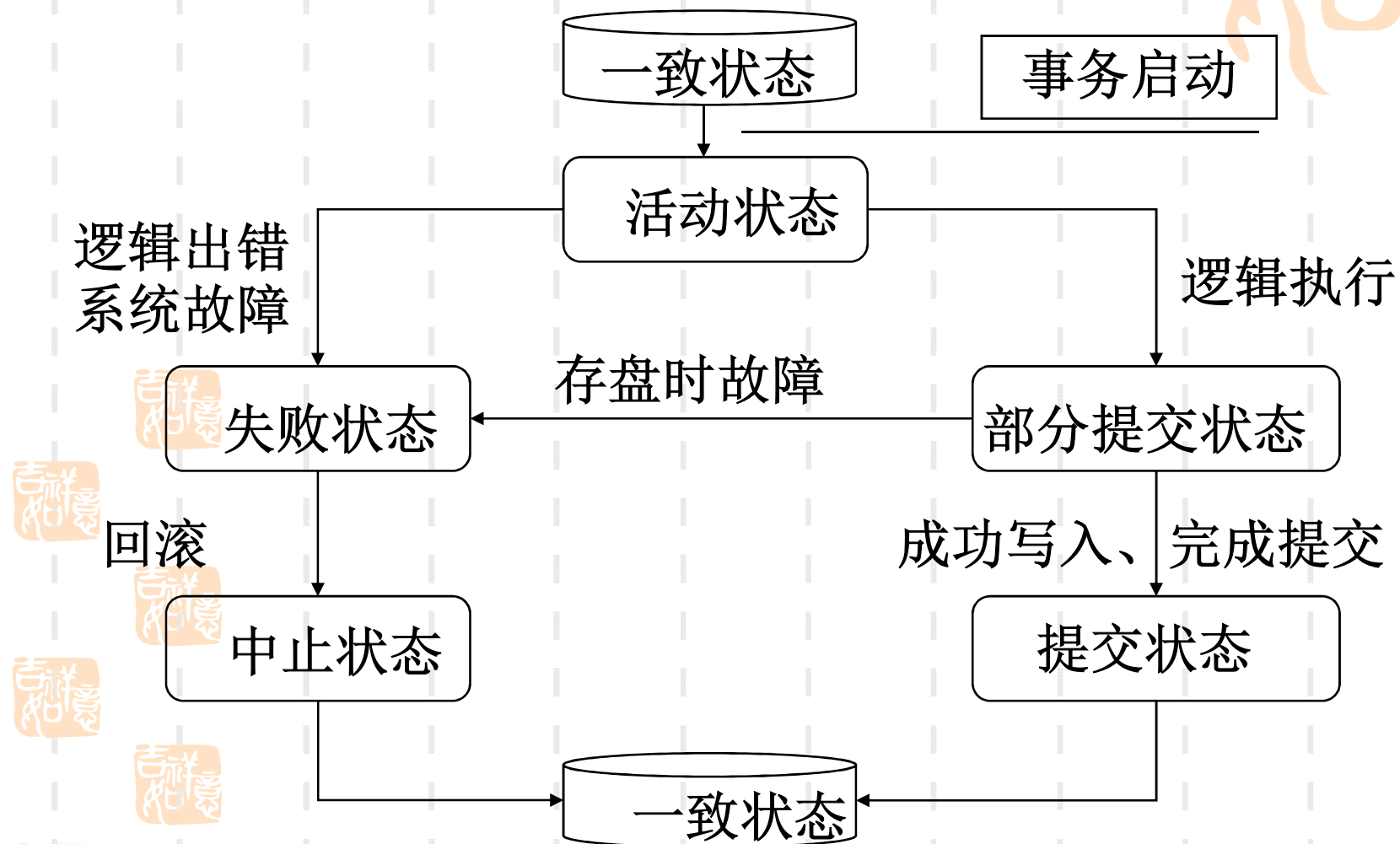
- 隔离性(Isolation):

- 一个事务的执行不能被其他事务干扰。

- 持续性(Durability):

- 一个事务提交后, 对数据库中的数据的改变是永久性的, 故障不应对其执行结果有影响。(即使在写入磁盘之前, 系统发生故障, 在下次启动之后, 也应保障数据更新的有效)

## 事务的状态



# 事务管理

- 恢复和并发控制是事务管理的重要组成部分
- **恢复管理部件**负责保证事务的原子性与持久性
- **并发控制部件**负责事务的并发控制机制，实现事务的隔离性与一致性
- 事务管理器实现事务的ACID

# 第十章 数据库恢复技术

10.1 事务的基本概念

10.2 数据库恢复概述

10.3 故障的种类

10.4 恢复的实现技术

10.5 恢复策略

10.6 具有检查点的恢复技术

10.7 数据库镜像

10.8 小结

## 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)

# 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

➤ 逻辑错误：一个事务由于其内部错误，导致不能正常结束（如事务内部的死循环）

➤ 系统错误：系统进入一个不良状态（如死锁），导致事务无法执行。但该事务在以后的某个时间可以重新执行。

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、系统断电等

- 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

磁盘上的物理数据或日志文件被破坏



# 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



## 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）
- 日志文件（logging）

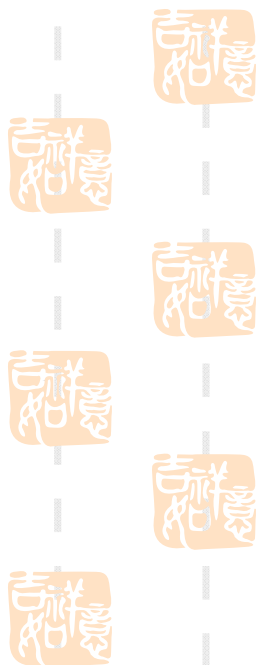
### 2. 如何利用这些冗余数据实施数据库恢复

## 10.4.1 数据转储



一、什么是数据转储

二、转储方法



# 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

## 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

## 10.4.2 登记日志文件



一、日志文件的格式和内容

二、日志文件的作用

三、日志文件



# 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
- 把写操作记录下来：
- 当事务 $T_i$ 开始时， $T_i$ 先在日志文件中写入如下的记录：

$\langle T_i \text{ start} \rangle$

- 当 $T_i$ 对记录 $X$ 执行写操作 $\text{write}(X)$ 时，首先写入日志记录 $\langle T_i, X, V_1, V_2 \rangle$ ，其中 $V_1$ 是上述写入操作前 $X$ 的值， $V_2$ 是新写入的值
- 当 $T_i$ 结束最后一条语句时，写入 $\langle T_i \text{ commit} \rangle$ 的日志记录
- 这里可以先假设日志记录是不经过缓存直接写到稳定的存储介质上的。

# Logging Example

- Fund transfer: A=50,C=100,B=50

Operation	Log Entry	Explanation
$R_1(A,50)$	(S,1)	Start Transaction $T_1$
$W_1(A,20)$	(W,1,A,50,20)	
$R_2(C,100)$	(S,2)	Start Transaction $T_2$
$W_2(C,50)$	(W,2,C,100,50)	
$C_2$	(C,2)	Commit $T_2$
$R_1(B,50)$	No log entry	
$W_1(B,80)$	(W,1,B,50,80)	
$C_1$	(C,1)	Commit $T_1$



# 日志文件的格式和内容（续）

- 把上述X取为一个物理块，则一个日志记录包含了如下三部分：

## 1. 前像（Before Image）：

当一个事务更新数据时，所涉及的物理块在更新前的映像称为该事务的前像，可以据此使数据库恢复到更新前的状态（撤消更新undo）。

## 2. 后像（After Image）

当一个事务更新数据时，所涉及的物理块在更新后的映像称为该事务的后像，可以据此使数据库恢复到更新后的状态（重做redo）。

## 3. 事务状态

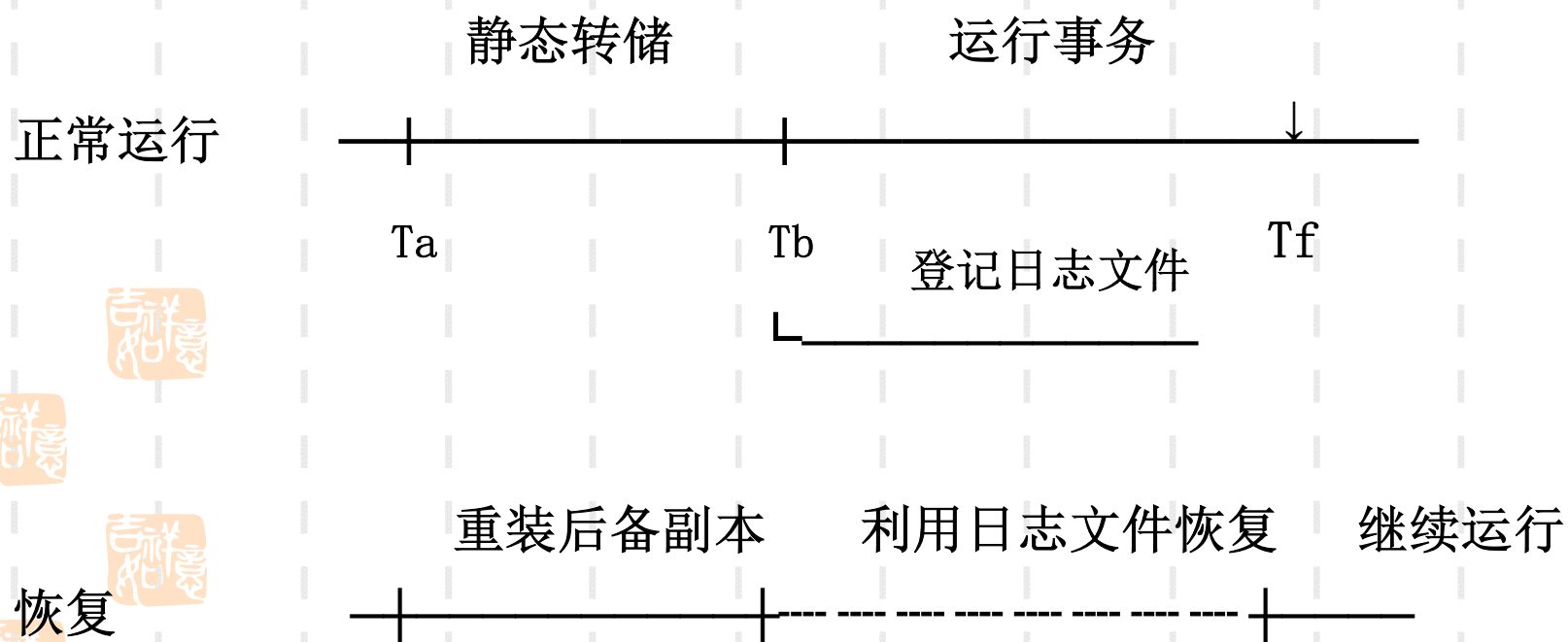
成功（committed）/失败（rollback, abort）；辅助的结构有活动事务表和提交事务表

## 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



### 三、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

# 第十章 数据库恢复技术

10.1 事务的基本概念

10.2 数据库恢复概述

10.3 故障的种类

10.4 恢复的实现技术

10.5 恢复策略

10.6 具有检查点的恢复技术

10.7 数据库镜像

10.8 小结

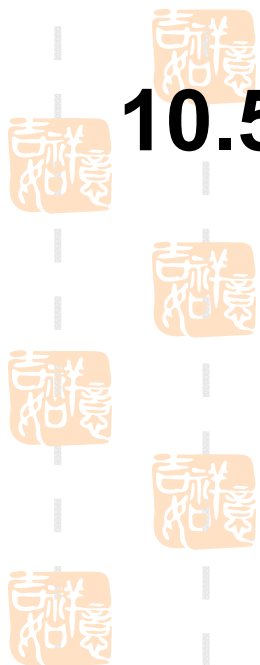
## 10.5 恢复策略



### 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



## 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

# Undo/Redo Logging Example

- Fund transfer: A=50, C=100, B=50
- crashed just after  $W_1(B, 80)$

Operation	Log Entry	Explanation
$R_1(A, 50)$	(S,1)	Start Transaction $T_1$
$W_1(A, 20)$	(W,1,A,50,20)	
$R_2(C, 100)$	(S,2)	Start Transaction $T_2$
$W_2(C, 50)$	(W,2,C,100,50)	
$C_2$	(C,2)	Commit $T_2$
$R_1(B, 50)$	No log entry	
$W_1(B, 80)$	(W,1,B,50,80)	
$C_1$	(C,1)	Commit $T_1$

# Recovery Example: Undo Phase

- **ROLLBACK** process,.

Log Entry	Explanation
1. (C,2)	Put $T_2$ into the committed list
2. (W,2,C,100,50)	Do nothing
3. (S,2)	Make a note that $T_2$ is not active
4. (W,1,A,50,20)	<b>UNDO</b> this update by writing the image value (50) into data item A. Put $T_1$ into the uncommitted list
5. (S,1)	Make a note that $T_1$ is not active

# Recovery Example: Redo Phrase

- ROLL FORWARD Process.

Log Entry	Explanation
6. (S,1)	No action required
7. (W,1,A,50,20)	No action required
8. (S,2)	No action required
9. (W,2,C,100,50)	<b>REDO</b> this update by writing after image value (50) into data item C.
10. (C,2)	No action required

Now A=?, B=?, C=?

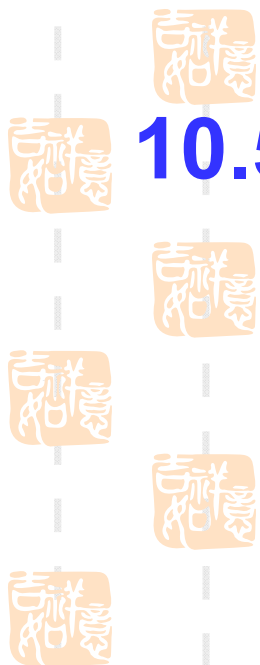
## 10.5 恢复策略



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

10.1 事务的基本概念

10.2 数据库恢复概述

10.3 故障的种类

10.4 恢复的实现技术

10.5 恢复策略

10.6 具有检查点的恢复技术

10.7 数据库镜像

10.8 小结

# 10.6 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



# 解决方案

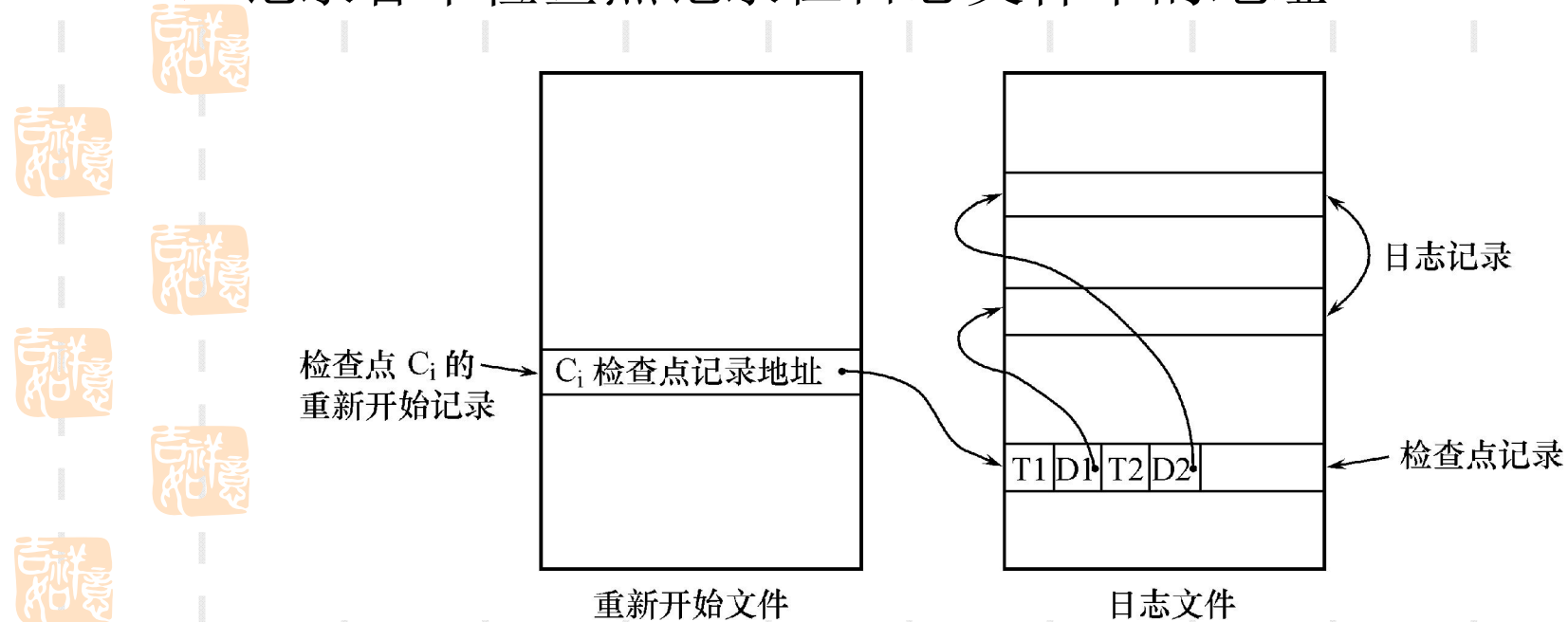
- 具有检查点（**checkpoint**）的恢复技术
  - 在日志文件中增加检查点记录（**checkpoint**）
  - 增加重新开始文件





## 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

- 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

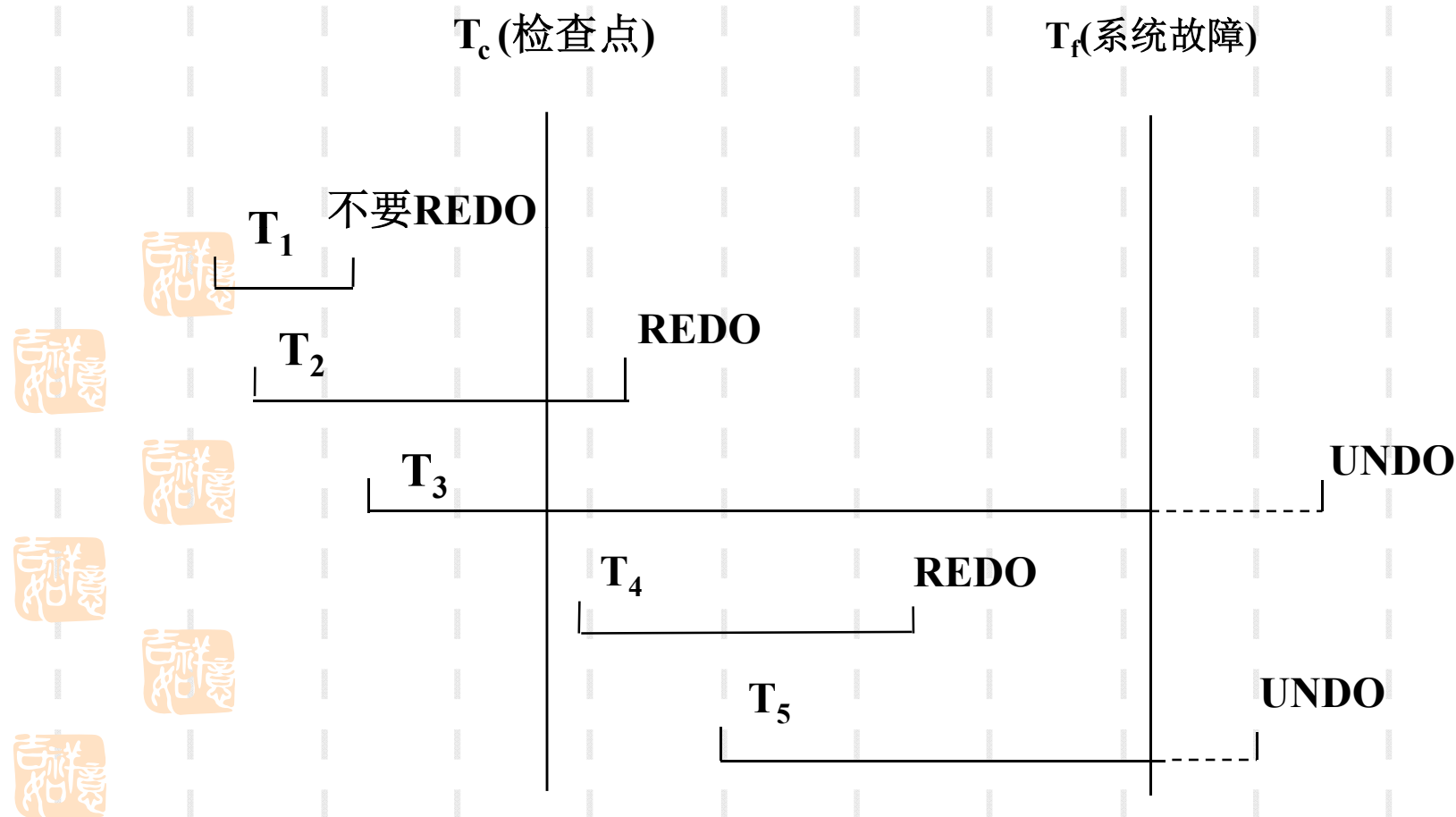
▶ 按照某种规则, 如日志文件已写满一半建立一个检查点

### 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
  - 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



# 第十章 数据库恢复技术

10.1 事务的基本概念

10.2 数据库恢复概述

10.3 故障的种类

10.4 恢复的实现技术

10.5 恢复策略

10.6 具有检查点的恢复技术

10.7 数据库镜像

10.8 小结

## 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

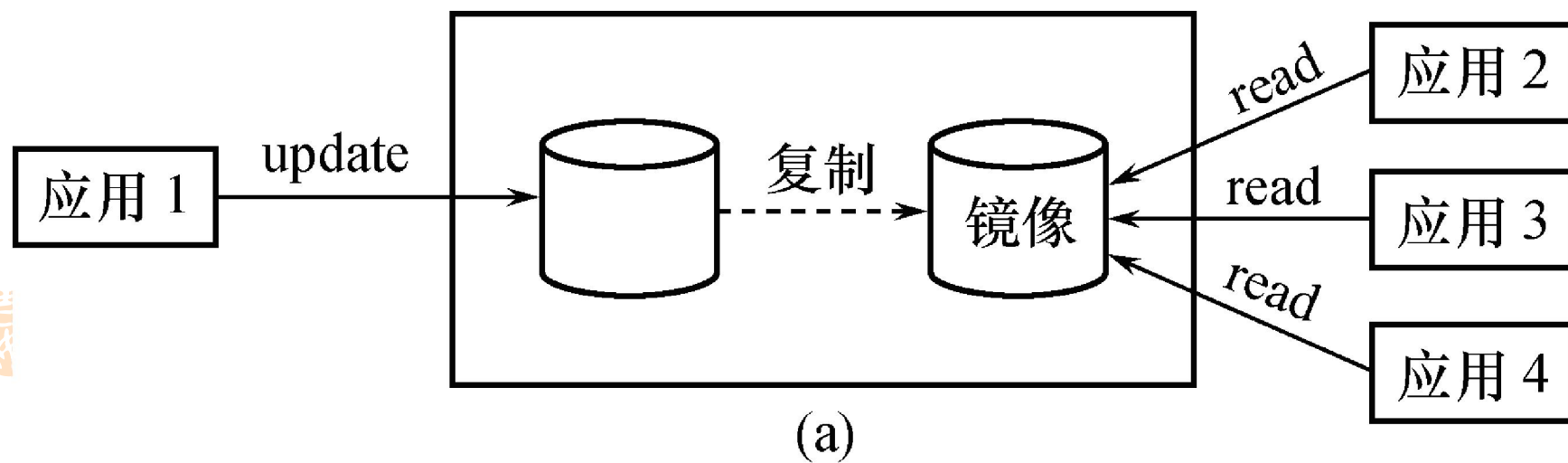


- 每当主数据库更新时，DBMS 自动把更新后的数据复制过去



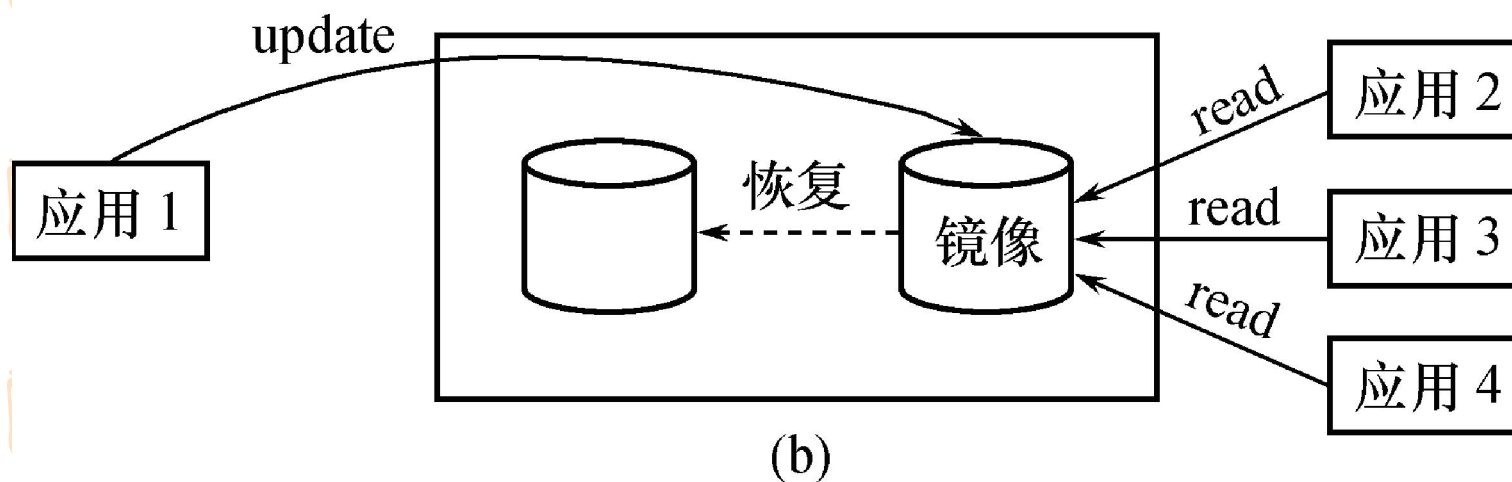


## 数据库镜像（续）



# 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



# 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



# 第十章 数据库恢复技术

10.1 事务的基本概念

10.2 数据库恢复概述

10.3 故障的种类

10.4 恢复的实现技术

10.5 恢复策略

10.6 具有检查点的恢复技术

10.7 数据库镜像

10.8 小结

## 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）

- 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

# 作业

■ 1, 4, 6

○

