第8章 数据库恢复技术

北京理工大学 计算机学院
张文耀
zhwenyao@bit.edu.cn

主要内容

- 8.1事务的基本概念和特征
- 8.2数据库恢复的必要性
- 8.4数据转储与恢复
- 8.5基于日志的数据库恢复
- 8.3数据库恢复策略
- 8.6检查点恢复技术
- 8.7数据库镜像恢复技术
- 8.8SQL Server的数据恢复机制
- 8.9小结

8.1 事务的基本概念和特征

- 数据库恢复技术是一种事务处理技术。
- 什么是事务(Transaction)？
  - 事务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。
  - 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。
  - 事务的开始和结束可以由用户显式控制。如果没有显式定义，则由DBMS按缺省规定自动划分事务。
- 事务处理技术主要包括数据库恢复技术和并发控制技术。

【例】银行转帐事务：从帐号A转1000元到帐号B

```
T: read(A);
   A:=A-1000;
   write(A);
   read(B);
   B:=B+1000;
   write(B);
```

上述转帐操作应该视为一个整体，要么全做，要么都不做；否则会出问题。

- 如何定义事务？

BEGIN TRANSACTION　　**BEGIN TRANSACTION**

SQL 语句　　**SQL 语句**

……　　**……**

COMMIT　　**ROLLBACK**

- **COMMIT**表示提交，提交事务的所有操作
  - 事务正常结束，事务中所有操作结果被保存，即：事务中所有对数据库的更新永久生效；
- **ROLLBACK**表示回滚，撤销事务的所有已完成的操作
  - 事务运行过程中出现故障，事务异常终止；
  - 系统撤销已完成的更新操作（回滚事务的所有更新操作），恢复到事务开始时的状态。

【例】银行转帐事务的定义

```
T:  BEGIN TRANSACTION
        read(A);
        A:=A-1000;
        write(A);
        if (A<0)    ROLLBACK;
        else{
           read(B);
           B:=B+1000;
           write(B);
           COMMIT;}
```

事务的基本特征

- 为了保证数据库中数据的正确性，要求事务具备**ACID特性**，即：
  - 原子性（**Atomicity**）
  - 一致性（**Consistency**）
  - 隔离性（**Isolation**）
  - 持久性（**Durability** ）
- 原子性
  - 一个事务对数据库的所有操作，在逻辑上是一个不可分割的工作单位。
  - 事务中的所有操作要么都执行，要么都不执行。
  - **DBMS**的事务管理子系统负责实现事务的原子性。

- 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执行的事务所提交的结果
  - 不一致状态：数据库中包含失败事务的执行结果
  - 一致性要求数据库中的数据不会因为事务的执行而遭受破坏，导致不正确的结果。
  - 示例：银行转帐事务T
    - 全做或者全不做，数据库都处于一致性状态。
    - 如果只做部分操作，数据库可能处于不一致性状态。

- 隔离性
  - 是指一个事务的执行不能影响到另一个事务，即一个事务的内部操作相对于外部事务是隔离的。
  - 多个事务并发执行时，各个事务之间不能互相干扰。
  - 隔离性的目的是：保证多个事务并发执行时，执行结果是正确的，多个事务的并发执行结果应该与这些事务先后单独执行的结果一样。
  - 隔离性由DBMS的并发控制子系统实现。

【例】隔离性被破坏的示例

t	T1	T2
1	Read(A)	
2		Read(A)
3	A:=A-1	
4	Write(A)	A:=A+2
5		Write(A)

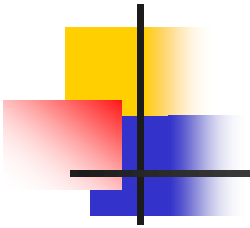
A表示某商品的库存
A的初值为10。

T1表示出库1件商品

T2表示入库2件商品

最后库存量应是11件

T1和T2的操作结果为12件，不符合现实。
原因在于：T1的操作结果被T2覆盖了！

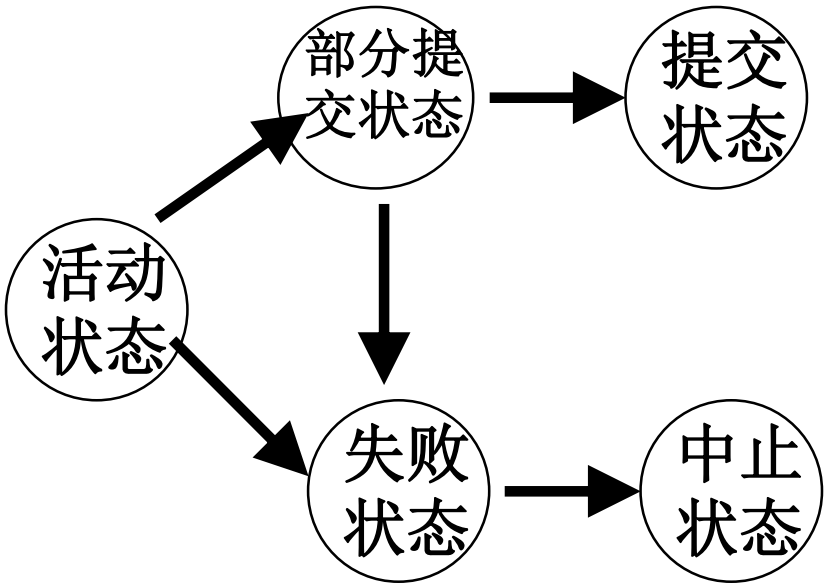


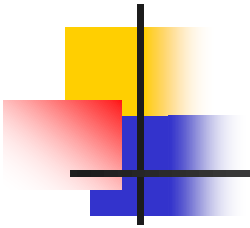
- 持久性
  - 持久性也称永久性（**Permanence**）
  - 一个事务一旦提交，它对数据库的所有更新应永久地反映在数据库中，不会丢失。接下来的其他操作或故障不应该对其执行结果有任何影响。
  - 持久性由**DBMS**的恢复管理子系统实现。

- 如果事务的**ACID**特性被破坏，会导致数据库中的数据不正确；因此保证事务的**ACID**特性是事务处理的重要任务。

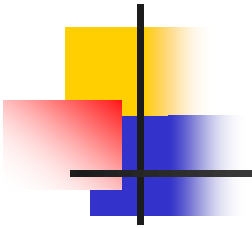
事务的状态

- 事务从开始到结束，需要经历一系列的状态变迁

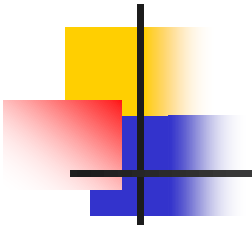




- 活动状态（active）
  - 初始状态
  - 事务开始执行后，立即进入活动状态
  - 在活动状态，事务将执行对数据库的读写操作；写操作的结果可能暂时存放在系统缓冲区
- 部分提交状态（partially committed）
  - 事务最后一条语句被执行后，进入部分提交状态
  - 事务执行完了，但对数据库的修改，很可能仍留在系统缓冲区中，不能说事务真正结束了。
  - 在此状态下，系统出现故障，仍可能使事务结果丢失，而导致事务失败。

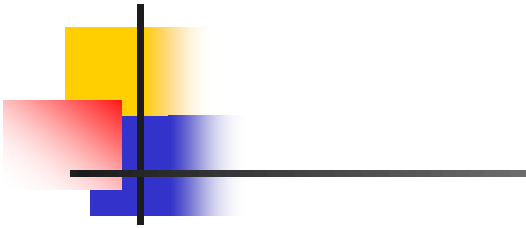


- 失败状态（**failed**）
  - 处于活动状态的事务还没有达到最后一个语句就中止执行，此时称此事务进入失败状态。
  - 处于部分提交的事务遇到故障也进入失败状态。
  - 处于失败状态的事务很可能对数据库中的数据做了部分修改（永久性的修改）
  - 失败状态意味着事务没有成功地完成，为了保证事务的原子性，应该撤销该事务对数据库已做的修改。
  - 对事务的撤销操作称为事务的回滚（**ROLLBACK**），即撤销事务已经做出的任何数据更改。



- 中止状态（**aborted**）
  - 夭折状态
  - 事务的异常结束状态
  - 处于失败状态的事务被回滚之后进入中止状态，此时数据库已经恢复到事务执行前的状态。

- 提交状态（committed）
  - 事务正常结束的状态
  - 事务进入部分提交状态后，系统执行提交（COMMIT）操作，把事务中对数据库的修改全部写入磁盘，并通知系统事务成功结束（写入一条事务日志信息，标志着事务成功完成），这时事务就进入了提交状态。



- 事务状态变迁中的相关概念
  - 事务中止
  - 事务回滚
  - 事务提交
- 事务状态变迁的原因
  - 当事务不能正常执行时，进入失败状态（**failed**）；
  - 失败状态的事务，回滚后，进入中止状态（**aborted**）；
  - 事务正常执行，并被成功提交之后，进入提交状态（**committed**）。

8.2 数据库恢复的必要性

- 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、恶意的破坏等
- 故障的影响
  - 事务非正常中断、事务的原子性遭到破坏，有可能丢失信息，破坏数据库的一致性
- 故障的对策
  - **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
  - 保证事务**ACID**
- 恢复技术是衡量数据库系统优劣的重要指标

- 数据库系统可能发生各种各样的故障，大致可分为：
  - 系统故障
  - 事务内部的故障
  - 存储设备故障
  - 其他原因导致的故障
    - 某些恶意的人为破坏，如病毒、恶意流氓软件等，造成事务异常结束。

- 什么是系统故障
  - 造成数据库系统停止运转的任何事件。
    - 硬件故障
    - 操作系统故障
    - **DBMS**故障
    - 突然停电
    - 操作员操作失误等
  - 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不受影响

- 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终止点就夭折了
- 事务内部的故障
  - 有的是可预期的(可以通过事务程序本身发现的)
    - 转账事务的例子
  - 有的是非预期的（不能由应用程序处理的）
    - 事务内部更多的故障是非预期的。
    - 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、并发事务发生死锁等
  - 以后，事务故障仅指非预期的故障。

- 存储设备故障（介质故障）
  - 辅助存储设备的存储介质遭到破坏
    - 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
  - 存储在外存中的数据部分丢失或全部丢失
  - 相对于前两类故障，介质故障的可能性比较小，但破坏性很大

- 其他原因的故障
  - 某些恶意的人为破坏，造成事务异常结束。如病毒，恶意流氓软件等

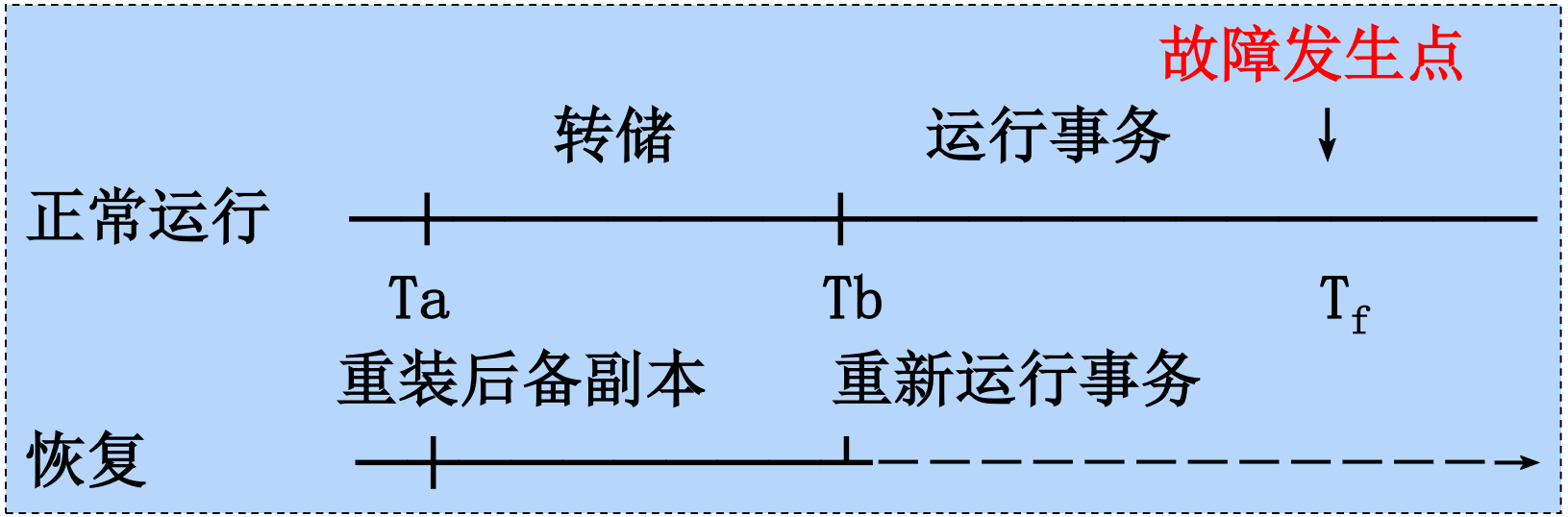
- 各类故障，对数据库的影响有两种可能性
  - 数据库本身被破坏（不能用了）
  - 数据库没有被破坏（可用），但数据可能不正确，这是由于事务的运行被非正常终止造成的。

对于各类故障，需要采取措施，尽可能恢复数据。

- 数据恢复的基本原理（方法）
  ——数据冗余
  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
  2. 如何利用这些冗余数据实施数据库恢复
- 建立冗余数据最常用的技术是
  - 数据转储和登记日志文件
  - 通常在一个数据库系统中，这两种方法一起使用

8.4 数据转储与恢复

- 数据转储是数据库恢复中采用的基本技术
  - DBA定期地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
  - 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



- 转储方法

1. 静态转储与动态转储
2. 海量转储与增量转储

		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

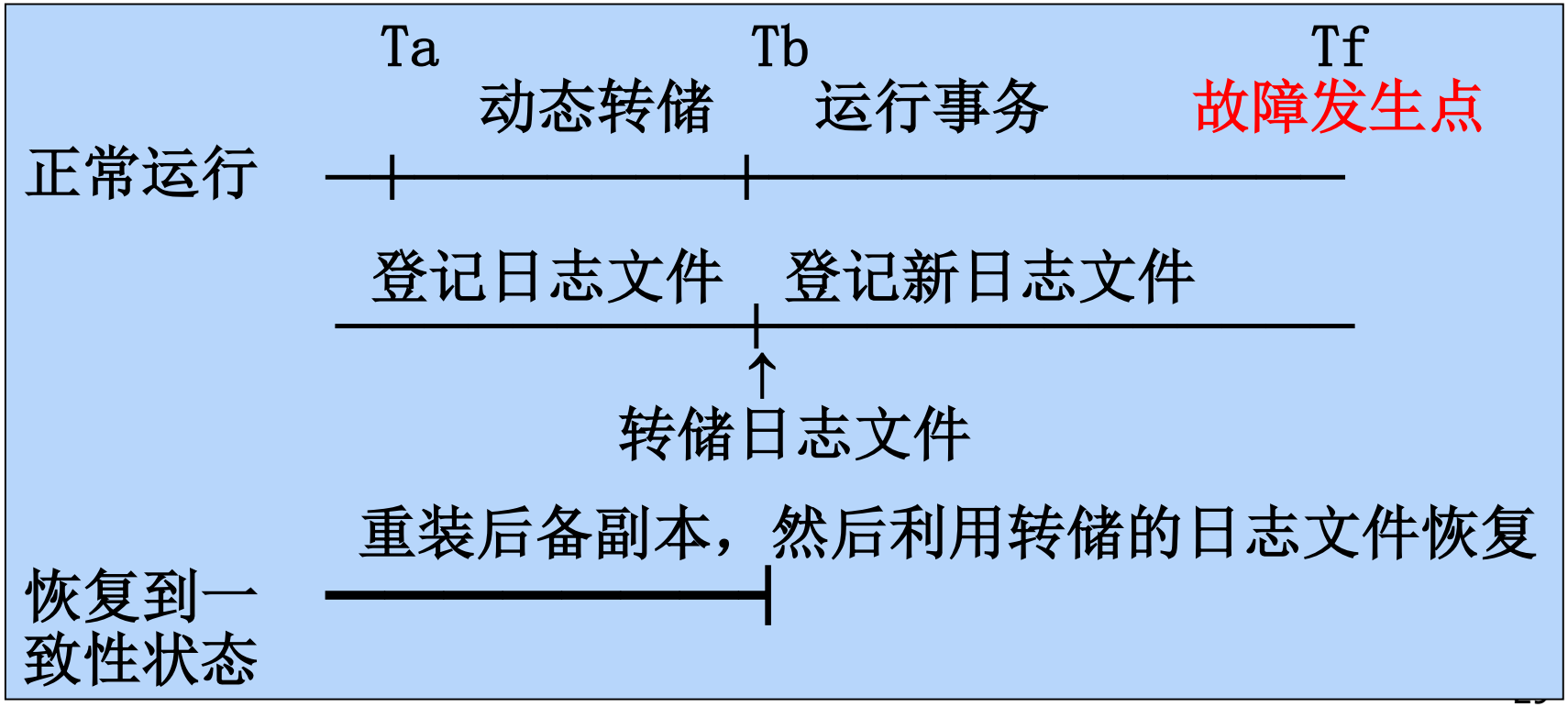
静态转储

- 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
- 缺点
  - 不能保证副本数据正确有效，需要配合日志进行恢复
  - 【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**过时

- 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



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

- 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、事务处理十分频繁，则增量转储方式更实用更有效。

转储的基本策略

- 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，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，例如：
  - 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

8.5 基于日志的数据库恢复

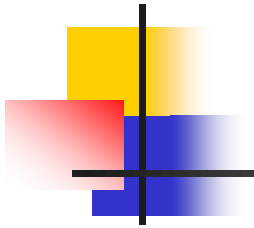
- 日志文件(日志，log)
  - 记录有关事务的数据库操作信息
- 日志文件的格式
  - 以记录为单位的日志文件
  - 以数据块为单位的日志文件
- 日志文件内容
  - 事务标识
  - 操作类型
  - 操作对象
  - 更新前后的数据值

- 以记录为单位的日志文件
  - **< Start *Ti*>**，表示事务***Ti***开始。
  - **<*Ti*， *Xj*， *V1*， *V2*>**，表示事务***Ti***对数据项***Xj***执行写操作，旧值是***V1***，新值是***V2***。
  - **< Commit *Ti*>**，表示事务***Ti***已提交。事务对数据库所做的任何更新都写入到数据缓冲区中，通常不能确定磁盘是否已经进行更新。
  - **< Abort *Ti*>**，表示事务***Ti***已中止。表明事务不能成功完成。如果事务中止，系统将确保这一事务的更新不会对数据库造成影响。

- 日志文件的作用
  - 日志在数据库系统中具有多重功能
  - 数据库恢复是日志文件的重要功能之一
  - 利用日志文件可以恢复系统故障和事务内部的故障
  - 对于介质故障，可以联合后备副本和日志文件进行有效恢复

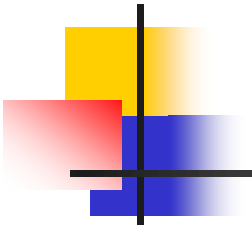
使用日志恢复数据库

- 基于日志的恢复技术：
  - **Redo**技术
    - 利用日志，重做已经提交的事务，把数据库恢复到故障前的状态。
    - 事务所有修改数据库的操作都记录在日志文件中，但是对数据库的修改可能还留在内存缓冲区中。
    - **Redo**操作数据项的顺序，与原来事务的操作顺序一致，是从头开始的。

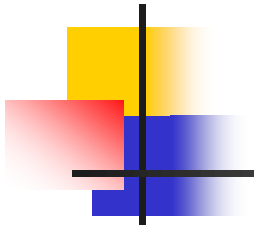


- **Undo技术**
  - 利用日志，撤销所有不可靠的修改。
  - 如果事务异常中止，事务对数据库的修改还没有提交，则需要利用**Undo**技术，撤销对数据库的修改，将数据库恢复到事务开始前的状态。
  - 检查日志文件，寻找事务**Ti**执行**write(X)**操作前写入日志的记录， 把数据库中的数据项**X**的值重新修改为更新前的旧值。
  - 如果事务**Ti**有多个**write**操作，**Undo**操作的顺序必须与**write**操作时写入日志的顺序相反。

- 基于日志的基本恢复策略
  - 对于故障发生时，已完成的事务进行**Redo** 处理
    事务***Ti***需要重新执行的条件是：日志中包括记录
    < Start *Ti*>和**< Commit *Ti*>**
  - 对于故障发生时，未完成的事务进行**Undo** 处理
    条件是：日志中只包含事务***Ti***的**< Start *Ti*>**记录，
    而没有**< Commit *Ti*>**记录。
  - 对于不同的故障类型，有不同的处理策略和实现过程



- 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
    - 写数据库和写日志文件是两个不同的操作，在此期间仍可能发生故障。
    - 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，以后就无法恢复这个修改。
    - 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/REDO**操作，并不会影响数据库的正确性。



- 基于日志的数据库恢复示例
  教材**P169 －171**

8.3 数据库恢复策略

- 应根据故障的类型，采取相应的恢复策略
  - 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中止
- 恢复方法：利用日志文件撤销事务对数据的更改，系统回滚到事务执行前的状态
- 事务故障的恢复由系统自动完成：

  (1) 反向扫描日志文件，查找该事务的更新操作。

  (2) 对该事务的更新操作进行**Undo**处理，即将日志记录中更新前的旧值，重新写入数据库。

  (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
  - 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没来得及写入数据库
- 恢复方法
  - **Undo** 未完成的事务
  - **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。

- 恢复步骤

(1) 正向扫描日志文件（即从头扫描日志文件），建立两个队列：

- **Redo**队列：记录在故障发生前已经提交的事务
- **Undo**队列：记录故障发生时尚未完成的事务

(2) 对**Undo**队列中的事务进行**Undo**处理

- 反向扫描日志文件，对每个需**Undo**的事务的更新操作执行逆操作（**Undo**）

(3) 对**Redo**队列中的事务进行**REDO**处理

- 正向扫描日志文件，对每个需**Redo**的事务重新执行所记录的更新操作（**Redo**）

介质故障的恢复

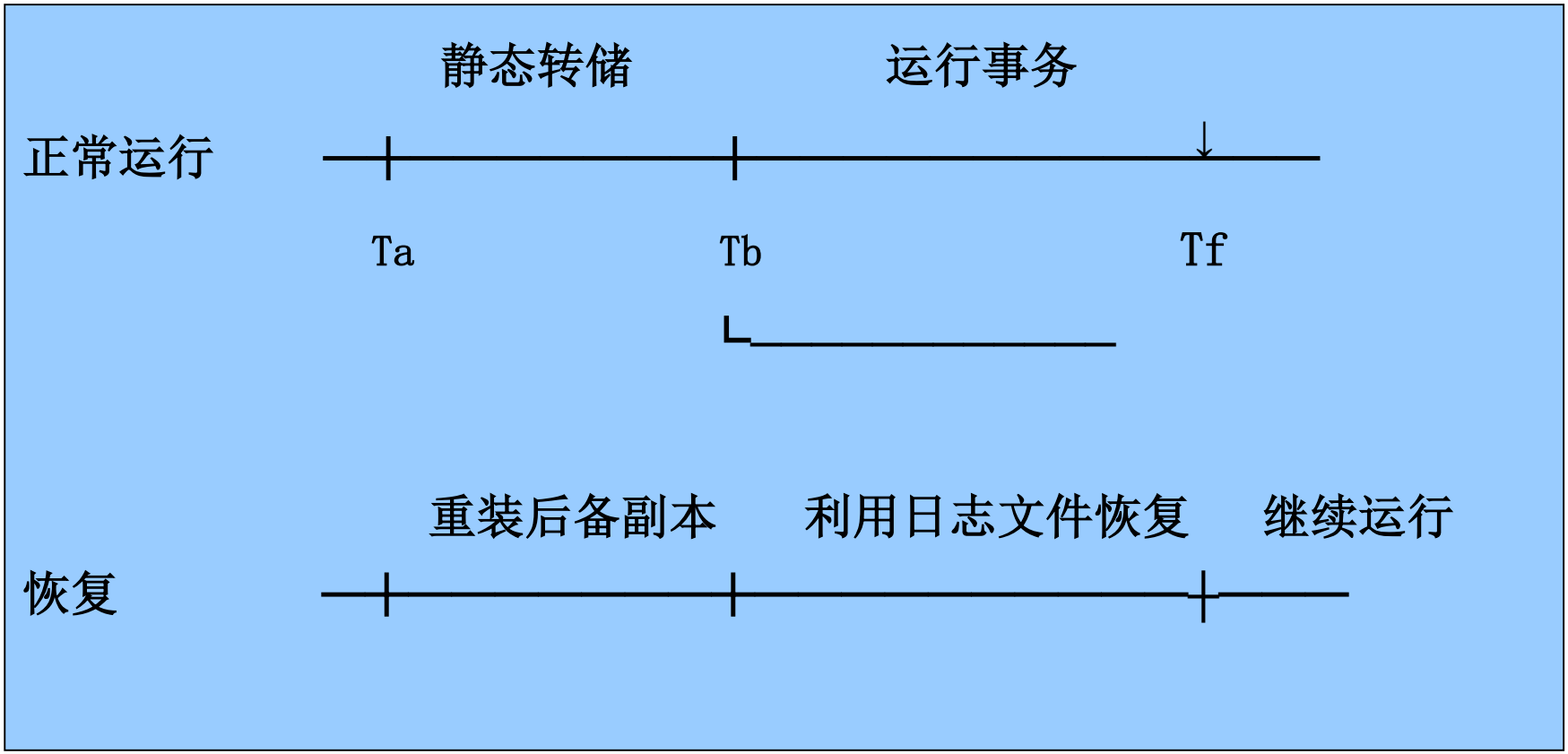
- 介质故障
  - 介质上存储的数据受到破坏
  - 单用日志文件不能恢复这些数据，需要后备副本
- 恢复方法
  - 重装数据库，使数据库恢复到某个一致性状态
  - 重做已完成的事务
- 介质故障的恢复需要**DBA介入**
  - 重装最近转储的数据库副本和相关日志文件副本
  - 执行系统提供的恢复命令
  - 具体的恢复操作仍由**DBMS**完成

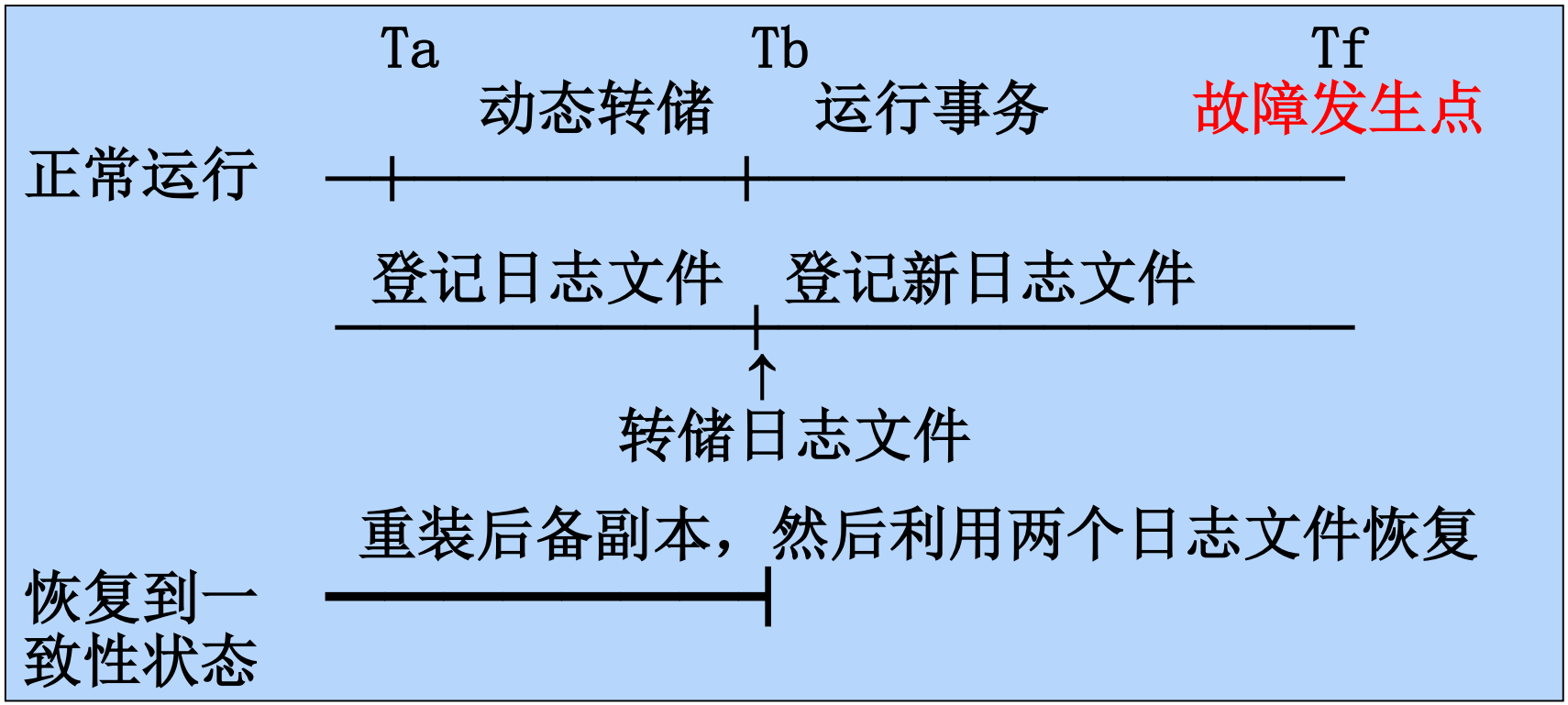
- 恢复步骤

(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处理日志文件，才能将数据库恢复到一致性状态。

(2) 装入有关的其他日志文件副本，重做已完成的事务（在数据库副本备份之后完成的事务）。


静态转储
运行事务
正常运行
Ta
Tb
Tf
重装后备副本
利用日志文件恢复
继续运行
恢复


Ta
Tb
Tf
动态转储
运行事务
故障发生点
正常运行
登记日志文件
登记新日志文件
转储日志文件
重装后备副本，然后利用两个日志文件恢复
恢复到一致性状态

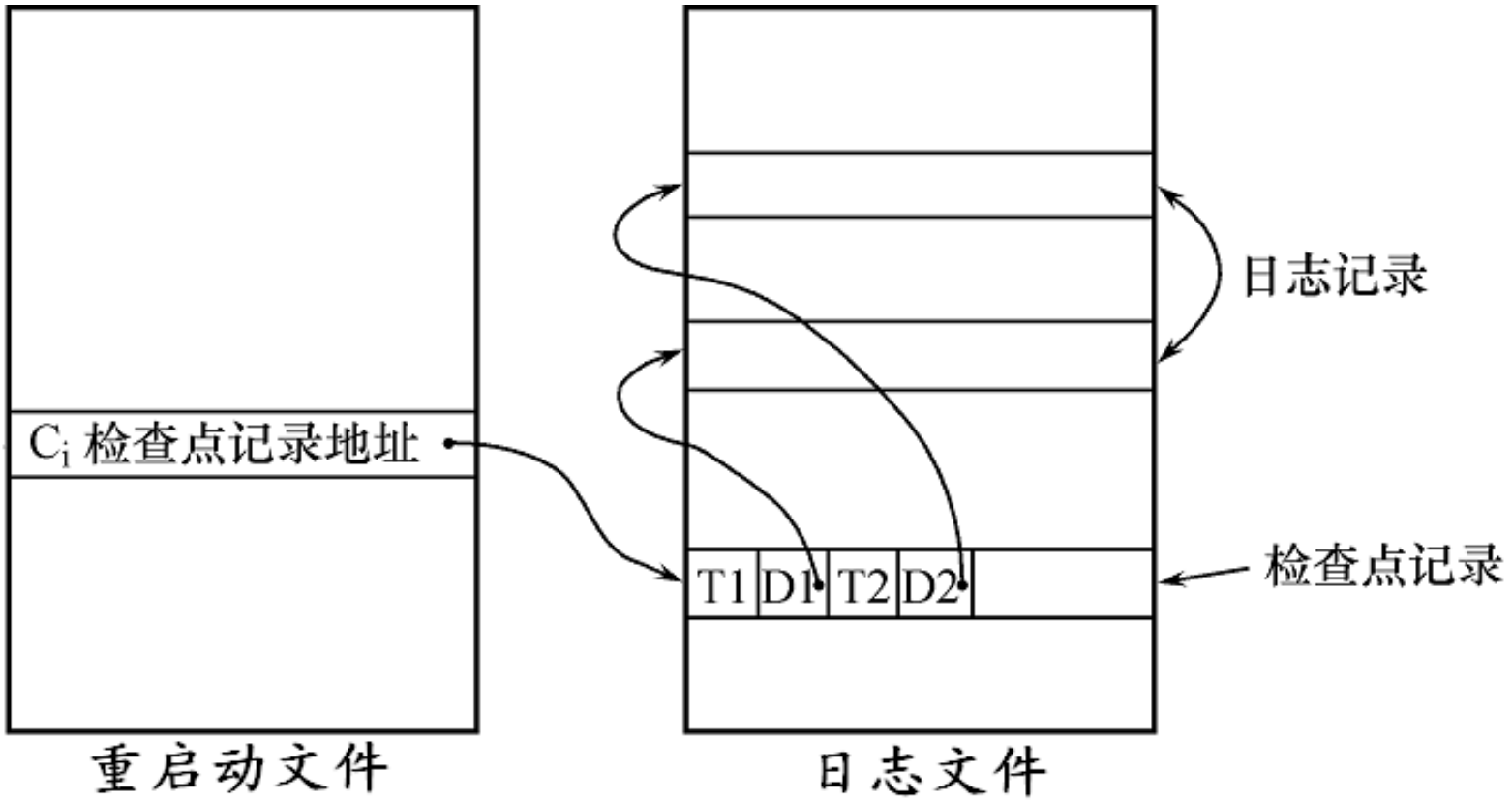
8.6 检查点恢复技术

- 问题的提出
  - 事务内部故障仅需恢复发生故障的个别事务
  - 系统故障影响到多个事务，恢复子系统需要扫描日志文件的所有记录，以便确定哪些事务需要**REDO**，哪些事务需要**UNDO**。
  - 搜索整个日志将耗费大量的时间，效率不高。
  - **REDO**操作也将浪费了大量时间，
    因为大部分事务更新实际已经写入数据库了，并不需要恢复。

- 为了提高恢复效率，提出了检查点恢复技术，允许恢复子系统在登记日志期间动态维护日志，为此：
  - 在日志文件中增加检查点记录
  - 增加重启动文件（重新开始文件）
- 检查点（**Checkpoint**）
  - 记录在日志中、表示数据库是否正常运行的一个标志，该标志记录所有当前活动事务的相关信息
- 检查点记录的内容
  - 建立检查点时所有正在执行的事务清单
  - 这些事务在日志中最近一个记录的地址

- 重启动文件

  记录各个“检查点记录”在日志文件中的地址
- 日志、检查点和重启动文件的关系



- 建立检查点
  - 写一个检查点，系统需要做以下事情

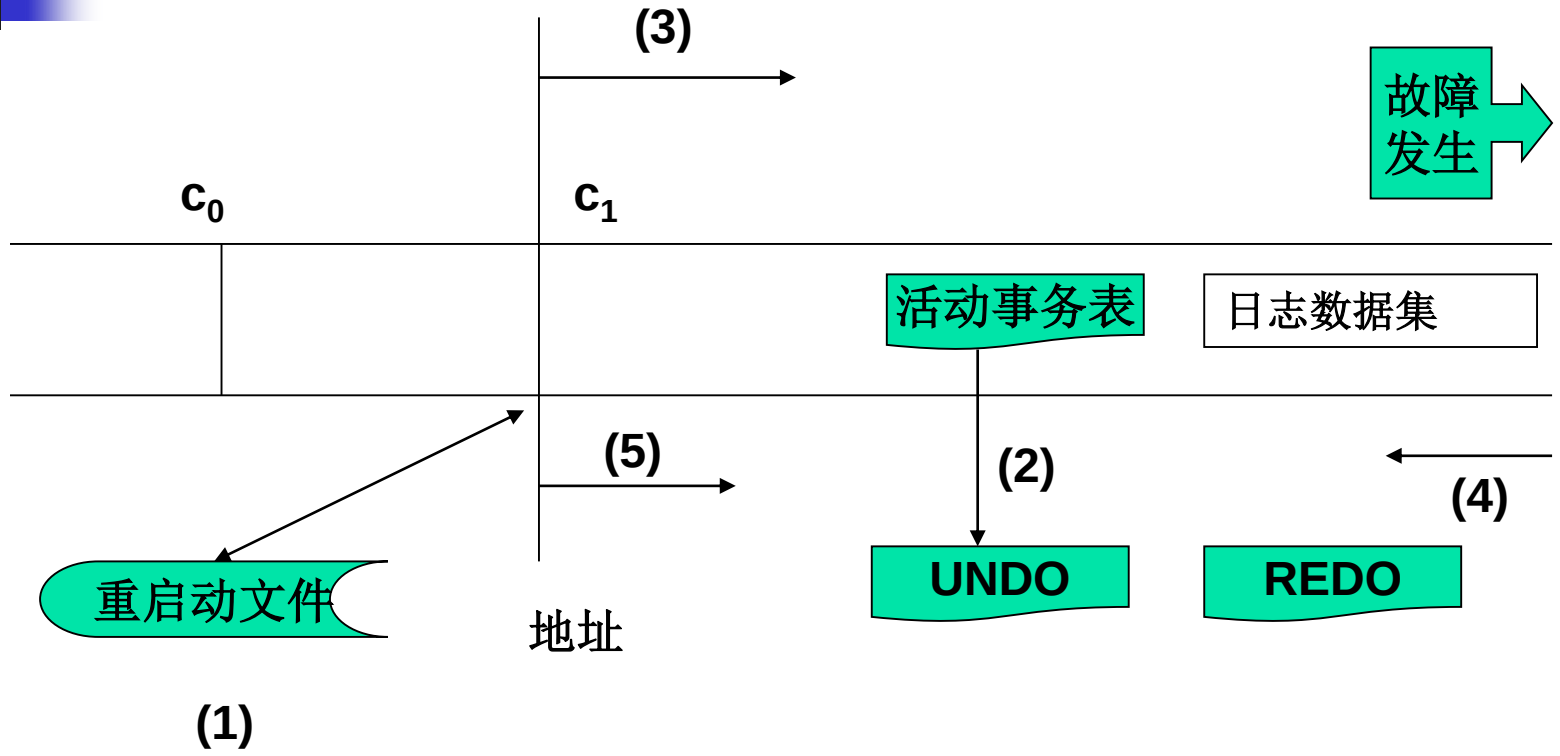
    （1）把日志缓冲区中的内容写入日志文件；

    （2）在日志文件中写入一个检查点记录；

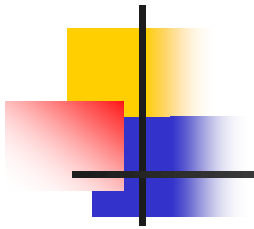
    （3）把数据库缓冲区的内容写入数据库；

    （4）把检查点记录在日志文件中的地址写入重启动文件
  - 写检查点步骤遵循“日志记录优先写入”的原则
  - 写检查点过程中，不允许事务执行任何更新操作

- 建立检查点原则
  - 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
- 利用检查点的恢复步骤

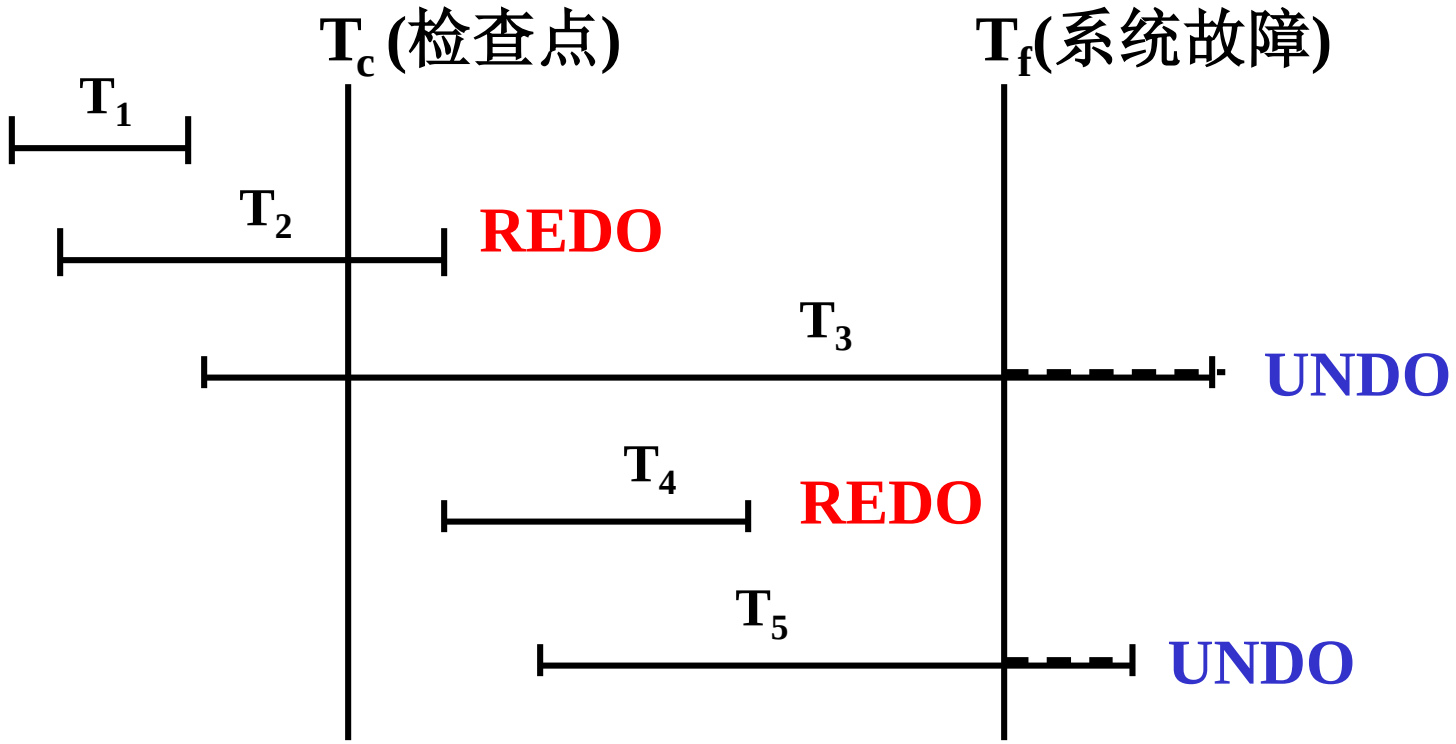


利用日志和检查点进行事务恢复的过程



1. 从“重启动文件” 读出最近检查点的地址，并定出检查点在日志文件中的 位置。
2. 创建**Redo**表(空)， **Undo**表(即检查点对应的活动事务表)。
3. 前向检索日志文件，如果遇到**Start**记录，则将对应的事务加入**Undo**表；如果遇到**Commit**的记录，则将该记录对应的事务从**Undo**表移到**Redo**表。
4. 反向检索日志文件， 对**Undo**表中的事务，按照日志记录，做**Undo**操作，直到遇到对应的**Start**记录。
5. 正向检索**Redo**事务的日志记录，做**Redo**操作，直到对应的**Commit**记录。

- 使用检查点方法可以改善数据库恢复效率
  - 如果事务**T**在一个检查点之前提交，在恢复时不需要重做，从而省却大量**REDO**操作。
    - 由写检查点记录的操作可知：事务**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
  - 只需针对检查点之后提交的事务，执行**REDO**操作
  - 由于检查点的存在，恢复时一般无需扫描整个日志文件

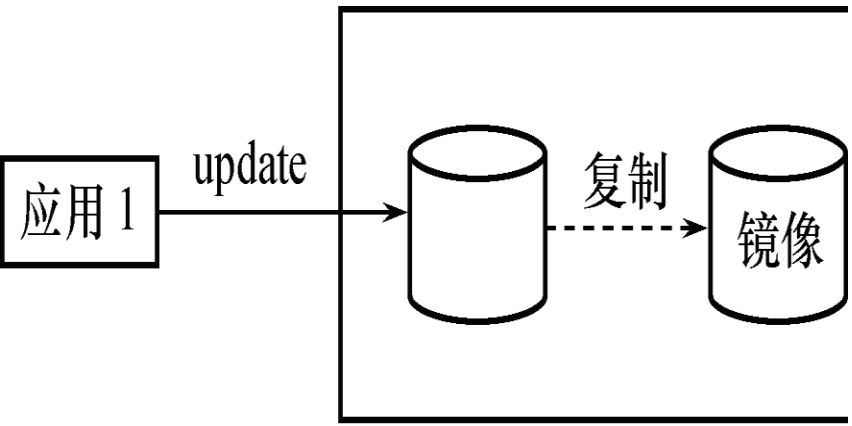


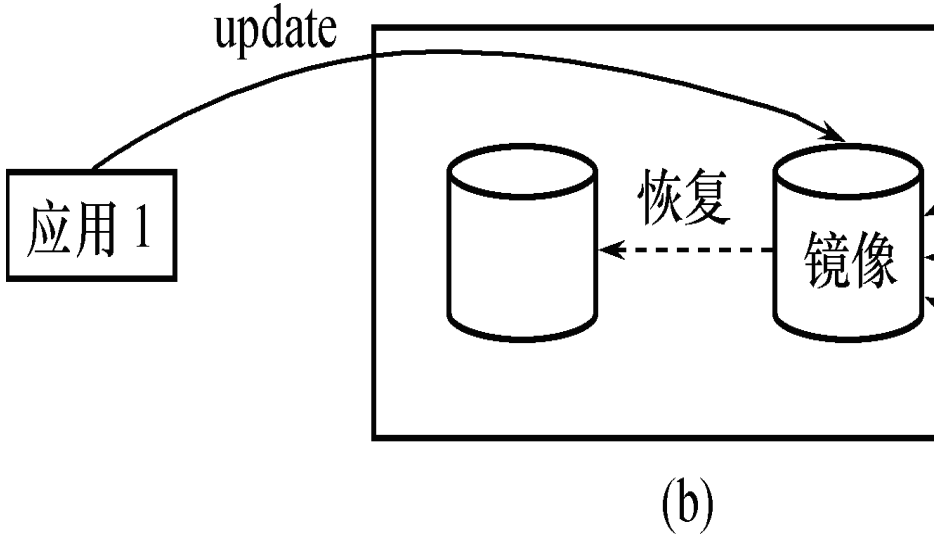
T_1 可以忽略 (因为有检查点，更新已经被写入磁盘)

8.7 数据库镜像恢复技术

- 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
  - 另一种应对介质故障的简单方法是数据库镜像
- 数据库镜像（**Mirror**）
  - **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
    - 每当主数据库更新时，**DBMS**自动把更新后的数据复制到数据库镜像中

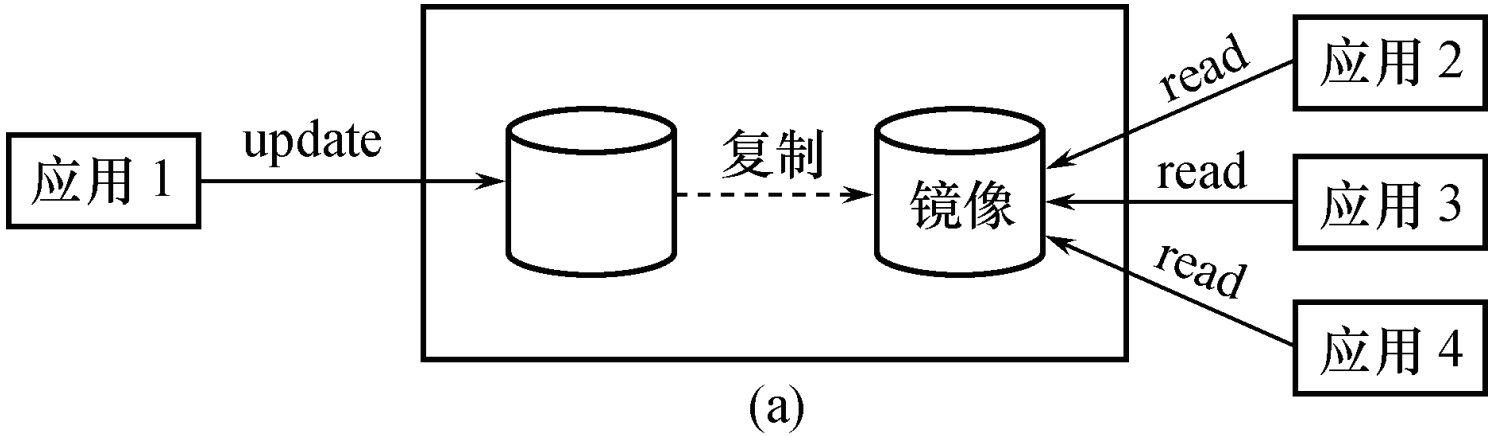
- 出现介质故障时
  - 可由镜像磁盘继续提供服务
  - 同时，**DBMS**自动利用镜像磁盘数据进行数据库的恢复
  - 不需要关闭系统和重装数据库副本，提高了数据库的可用性





(b)

- 没有出现故障时
  - 可用于并发操作，即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



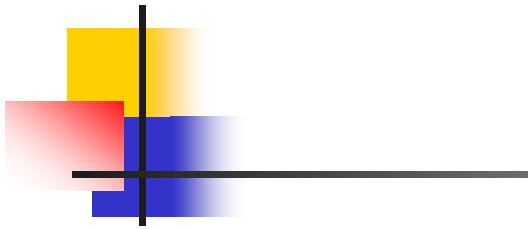
(a)

频繁地复制数据自然会降低系统运行效率

在实际应用中往往只选择对关键数据和日志文件进行镜像，而不是对整个数据库进行镜像

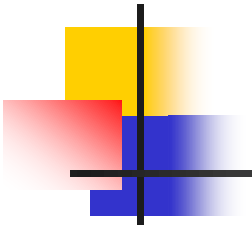
8.8 SQL Server的数据恢复机制

- **SQL Server的事务**
  - 一条语句
  - 一组语句或者是一个程序
  - **DBMS**保证事务的原子性
- **SQL Server事务运行模式**
  - 自动提交事务
    单独的**T-SQL**语句
  - 显式事务，用户定义或指定的事务
  - 隐式事务，每个**T-SQL**语句，如**INSERT**、**UPDATE**和**DELETE**等，都被作为一个事务执行。

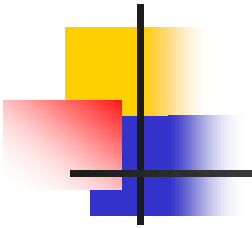


- **SQL Server的自定义事务**
  - **BEGIN TRANSACTION [t_name]**
  - **COMMIT [t_name]**
  - **ROLLBACK [t_name]**

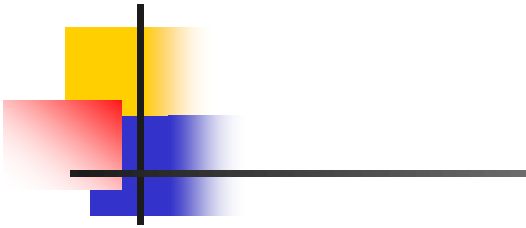
可以将多条**T－SQL**语句组装成一个事务



- **SQL Server的隐式事务**
  - 会自动启动新事务，无需描述事务的开始，只需要提交或回滚每个事务；在**COMMIT**或**ROLLBACK**之前，事务一直保持有效。
  - 新事务按默认规则启动，一个事务可能涉及多个**T-SQL**语句。
  - 在默认情况下是关闭的
  - 开启语句为：
    SET IMPLICIT_TRANSACTIONS ON



- **SQL Server**提供了一组备份和恢复操作，可以从系统和介质故障中恢复事务。
- **SQL Server**提供的备份方法
  - 完整备份
  - 差异备份
  - 事务日志备份
  - 数据库文件或者文件组备份

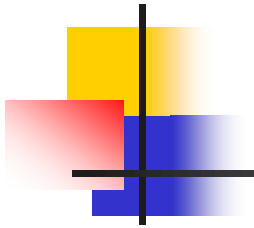


- 完整备份
  - 备份所有数据，以及可以恢复这些数据的足够多的日志
  - 虽然保证了数据的安全，但还原工作量太大
  - 如果数据库频繁修改，那么用差异备份可以达到同样的效果，但还原所需时间少

- 差异备份
  - 备份上一次完整备份后数据库中更改的部分
  - 要求一个完整的数据库备份，该备份称为差异基准
  - 备份内容仅包括自建立差异基准后发生更改的数据
  - 特点
    - 速度快，可在短时间内执行，可以节省备份和恢复过程的时间，适合于频繁修改数据的情况
    - 随着更改内容的增加，差异备份会逐渐增大；经过一段时间后，应重新创建一个完整备份，建立新的差异基准。

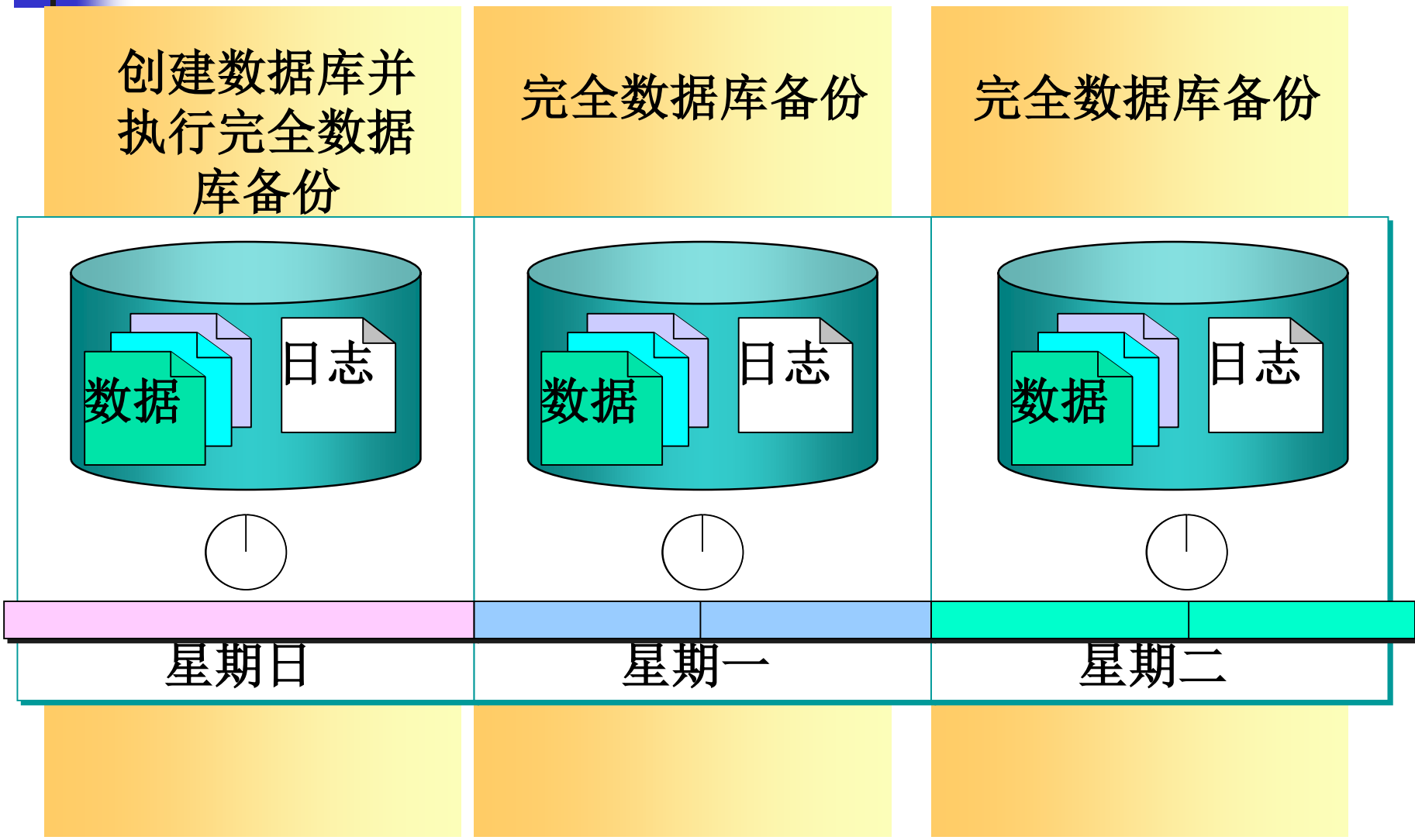
- 事务日志备份
  - 简称日志备份
  - 备份内容包括：创建备份时处于活动状态的部分事务日志，以及先前日志备份中未备份的所有日志记录
  - 使用事务日志备份可将数据库恢复到特定的时间点或恢复到故障点
  - 事务日志备份比数据库备份使用的资源少，可以比数据库备份更经常地创建事务日志备份。
  - 日志备份需结合完全恢复模型或大容量日志记录恢复模型一起使用

- 数据库文件或者文件组备份
  - 对一些超大型数据库进行完全数据库备份可能无法管理，此时可以只备份一部分文件
  - 文件备份操作必须与事务日志备份一起使用
  - 还原文件后，必须还原自创建文件备份后创建的事务日志备份以使数据库处于一致状态
  - 文件备份只适用于完全恢复模型和大容量日志记录恢复模型
  - 文件备份可以快速恢复已隔离的媒体故障，可以迅速还原损坏的文件
  - 具体操作和管理比较复杂，可参阅相关技术文档

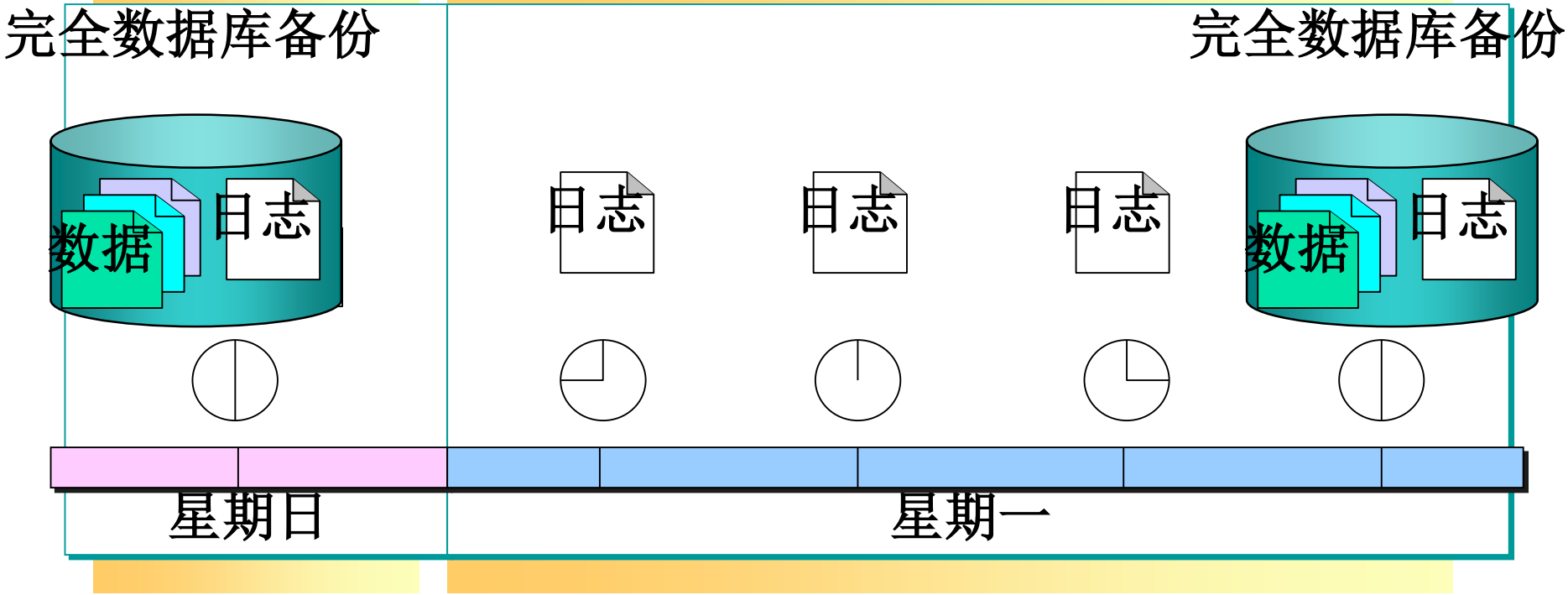


- **SQL Server 的数据库备份策略**
  - 完全数据库备份策略
  - 完全数据库备份＋事务日志备份策略
  - 完全数据库备份＋差异备份＋事务日志备份策略
  - 数据库文件或文件组备份＋日志备份策略

完全数据库备份策略


创建数据库并执行完全数据库备份
完全数据库备份
完全数据库备份
数据
日志
数据
日志
数据
日志
星期日
星期一
星期二

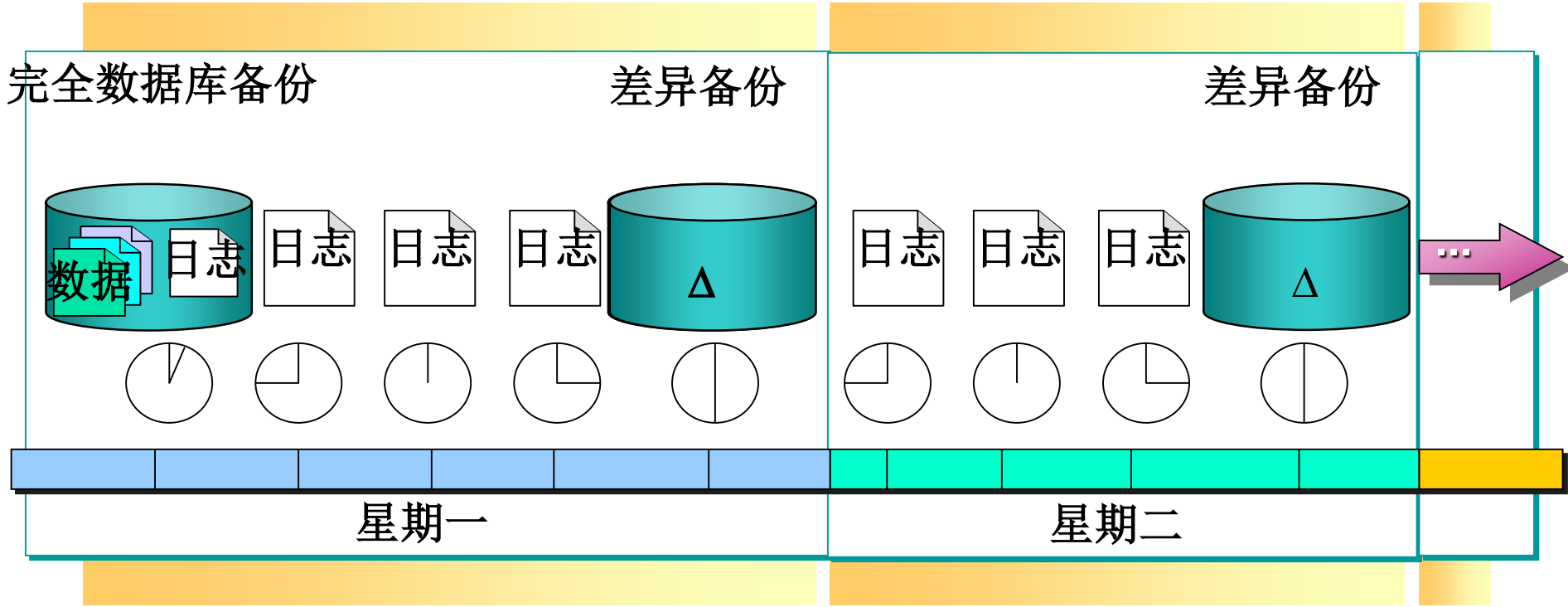
完全数据库＋事务日志备份策略



- 定期进行完全备份，如一天一次或两天一次。
- 更频繁地进行事务日志备份，如一小时一次。
- 当需要数据库恢复时，首先用最近一次完全备份恢复数据库，然后用最近一次完全备份之后创建的所有事务日志备份，按顺序恢复完全备份之后发生在数据库上的所有操作。
- 完全备份和事务日志备份相结合的方法，能够完成许多数据库的恢复工作。
- 对那些不在事务日志中留下记录的操作，仍无法恢复数据。

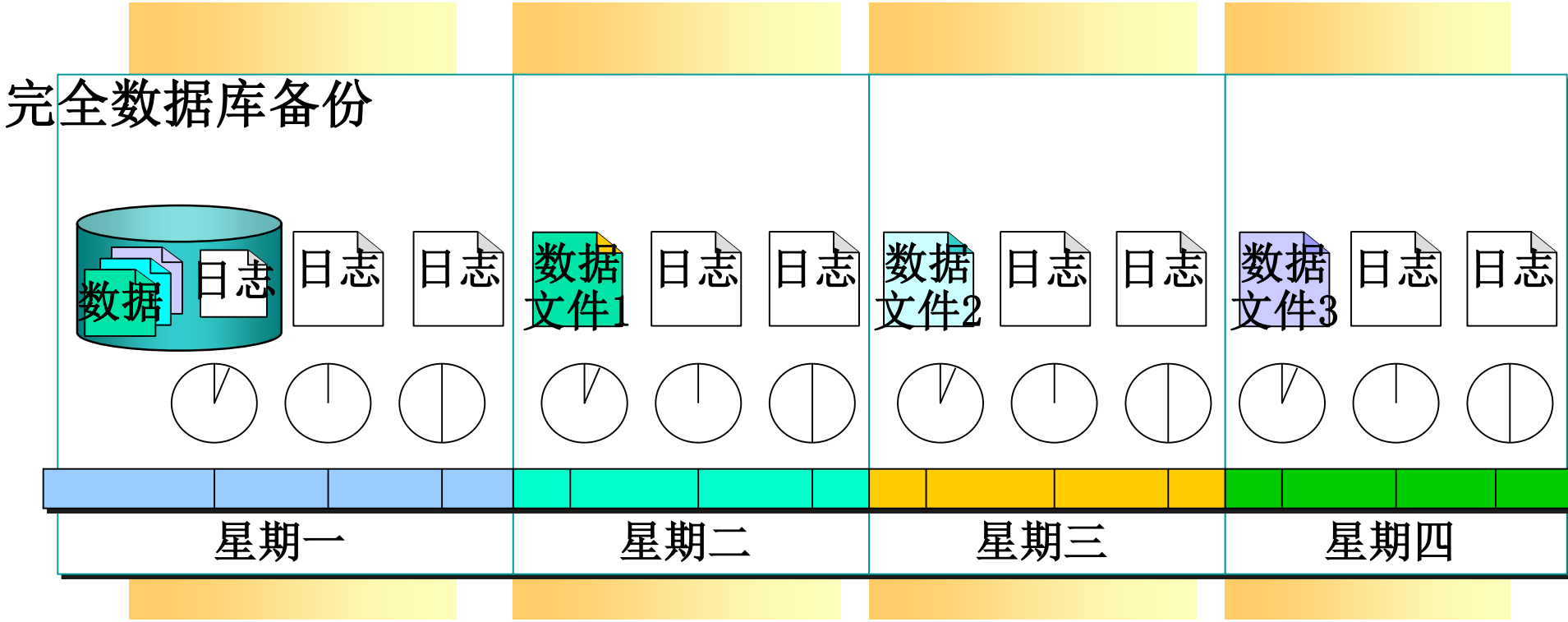
完整备份＋差异备份＋日志备份的策略

- 使用数据库备份、差异数据库备份和事务日志备份的组合，可以将故障恢复所需的时间减到最少



数据库文件组备份＋日志备份策略

- 适用于超大型数据库
- 必须包含完全数据库备份和事务日志备份



恢复模型

- 恢复模型与备份策略联系在一起
  在进行备份前，需要确定使用哪种恢复模型
- **SQL Server**提供的恢复模型
  - 简单恢复
    - 简单恢复允许将数据库恢复到最新的备份。
  - 完全恢复
    - 完全恢复允许将数据库恢复到故障点状态。
  - 大容量日志记录恢复
    - 大容量日志记录恢复允许大容量日志记录操作。

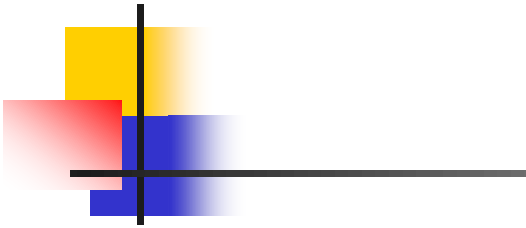
- 简单恢复模式
  - 提供简单的备份与还原
  - 不备份事务日志
  - 只能将数据还原到数据最近一次备份的结尾
  - 备份之后所做的修改将全部丢失
  - 可以通过提高备份频率减少丢失数据的风险
  - 过高的频率会影响性能，备份本身也难以管理

- 完整恢复模式
  - 使用日志备份
  - 允许将数据库还原到日志备份包含的任何时间点
  - 可在最大范围内防止数据丢失；数据丢失的风险仅在最新日志备份之后；
  - 出现故障后，应先尝试备份“日志尾部”－尚未备份的日志，以便将数据库还原到故障点，避免任何工作的丢失
  - 增加还原时间和还原的复杂性
  - 但是，大多数应用采用日志备份

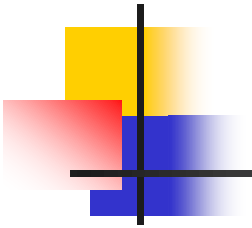
- 如何选择恢复模型和备份方法
  - 必须确定数据库中的数据可以承担的损失程度
  - 必须确定数据库在多长时间内不可用，是否必须尽快使数据库重新联机，或者是否不需要立即还原
  - 必须根据可用的资源要求来选择数据库的恢复模式
  - 尽量选择运行时对系统的影响最小，同时又能满足还原要求的备份过程

本章小结

- 事务的概念和特征
  - 事务是最小的不可再分的逻辑操作单元
  - 基本特征：**ACID**
- 故障是不可避免的
- 数据库必须具备恢复机制，以保证事务的原子性
- 恢复的基本原理
  - 利用存储在后备副本、日志文件或数据库镜像中的冗余数据来重建数据库



- 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**



- 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
- **SQL Server的数据恢复机制**

本章思考题

- 教材P180（e187）
  习题1，2，3，4，5，7，8