第10章 数据库恢复技术

本书第 10 章、第 11 章讨论**事务处理**（transaction processing）技术。事务是一系列的数据库操作，是数据库应用程序的基本逻辑单元。事务处理技术主要包括数据库恢复技术和并发控制技术。数据库恢复机制和并发控制机制是数据库管理系统的重要组成部分。本章讨论数据库恢复的概念和常用技术。

10.1 事务的基本概念

在讨论数据库恢复技术之前先讲解事务的基本概念和事务的性质。

1. 事务

所谓**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。例如，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。

事务和程序是两个概念。一般地讲，一个程序中包含多个事务。

事务的开始与结束可以由用户显式控制。如果用户没有显式地定义事务，则由数据库管理系统按默认规定自动划分事务。在 SQL 中，定义事务的语句一般有三条：

```
BEGIN TRANSACTION;
COMMIT;
ROLLBACK;
```

事务通常是以 BEGIN TRANSACTION 开始，以 COMMIT 或 ROLLBACK 结束。COMMIT 表示提交，即提交事务的所有操作。具体地说就是将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去，事务正常结束。ROLLBACK 表示回滚，即在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，回滚到事务开始时的状态。这里的操作指对数据库的更新操作。

2. 事务的 ACID 特性

事务具有 4 个特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持续性（Durability）。这 4 个特性简称为 ACID 特性（ACID properties）。

（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。

（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。因此当数据库只包含成功事务提交的结果时，就说数据库处于一致性状态。如果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，这些未完成的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，或者说是不一致的状态。例如，某公司在银行中有 A，B 两个账号，现在公司想从账号 A 中取出一万元，存入账号 B。那么就可以定义一个事务，该事务包括两个操作，第一个操作是从账号 A 中减去一万元，第二个操作是向账号 B 中加入一万元。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，则逻辑上就会发生错误，减少或增加一万元，这时数据库就处于不一致性状态了。可见一致性与原子性是密切相关的。

（3）隔离性

一个事务的执行不能被其他事务干扰。即一个事务的内部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。

（4）持续性

持续性也称**永久性**（Permanence），指一个事务一旦提交，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。

事务是恢复和并发控制的基本单位，所以下面的讨论均以事务为对象。

保证事务 ACID 特性是事务管理的重要任务。事务 ACID 特性可能遭到破坏的因素有：

（1）多个事务并行运行时，不同事务的操作交叉执行；

（2）事务在运行过程中被强行停止。

在第一种情况下，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原子性；在第二种情况下，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。

这些就是数据库管理系统中恢复机制和并发控制机制的责任。

10.2 数据库恢复概述

尽管数据库系统中采取了各种保护措施来防止数据库的安全性和完整性被破坏，保证

并发事务的正确执行，但是计算机系统中硬件的故障、软件的错误、操作员的失误以及恶意的破坏仍是不可避免的，这些故障轻则造成运行事务非正常中断，影响数据库中数据的正确性，重则破坏数据库，使数据库中全部或部分数据丢失。因此数据库管理系统必须具有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。恢复子系统是数据库管理系统的一个重要组成部分，而且还相当庞大，常常占整个系统代码的10%以上。数据库系统所采用的恢复技术是否行之有效，不仅对系统的可靠程度起着决定性作用，而且对系统的运行效率也有很大影响，是衡量系统性能优劣的重要指标。

10.3 故障的种类

数据库系统中可能发生各种各样的故障，大致可以分以下几类。

1. 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，不能由事务程序处理。

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*/
    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;}
```

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，例如可能出现只把账户甲的余额减少而没有把账户乙的余额增加的情况。

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。

事务内部更多的故障是非预期的，是不能由应用程序处理的。如运算溢出、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。本书后续内容中，事

务故障仅指这类非预期的故障。

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),因此,数据库可能处于不正确状态。恢复程序要在不影响其他事务运行的情况下,强行回滚该事务,即撤销该事务已经作出的任何对数据库的修改,使得该事务好像根本没有启动一样。这类恢复操作称为**事务撤销**(UNDO)。

2. 系统故障

系统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。例如,特定类型的硬件错误(CPU 故障)、操作系统故障、DBMS 代码错误、系统断电等。这类故障影响正在运行的所有事务,但不破坏数据库。此时主存内容,尤其是数据库缓冲区(在内存)中的内容都被丢失,所有运行事务都非正常终止。发生系统故障时,一些尚未完成的事务的结果可能已送入物理数据库,从而造成数据库可能处于不正确的状态。为保证数据一致性,需要清除这些事务对数据库的所有修改。

恢复子系统必须在系统重新启动时让所有非正常终止的事务回滚,强行撤销所有未完成事务。

另一方面,发生系统故障时,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,这也会使数据库处于不一致状态,因此应将这些事务已提交的结果重新写入数据库。所以系统重新启动后,恢复子系统除需要撤销所有未完成的事务外,还需要**重做**(REDO)所有已提交的事务,以将数据库真正恢复到一致状态。

3. 介质故障

系统故障常称为**软故障**(soft crash),介质故障称为**硬故障**(hard crash)。硬故障指外存故障,如磁盘损坏、磁头碰撞,瞬时强磁场干扰等。这类故障将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。这类故障比前两类故障发生的可能性小得多,但破坏性最大。

4. 计算机病毒

计算机病毒是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。这种程序与其他程序不同,它像微生物学所称的病毒一样可以繁殖和传播,并造成对计算机系统包括数据库的危害。

计算机病毒的种类很多,不同病毒有不同的特征。小的病毒只有 20 条指令,不到 50 B。大的病毒像一个操作系统,由上万条指令组成。

有的计算机病毒传播很快,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。多数病毒一开始并不摧毁整个计算机系统,它们可能只在数据库或其他数据文件中将小数点向左或向右移一两位,增加或删除一两个“0”,从而导

致系统运行不正常。

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。为此计算机的安全工作者已研制了许多预防病毒的“疫苗”，检查、诊断、消灭计算机病毒的软件也在不断发展。但是，至今还没有一种可以使计算机“终生”免疫的“疫苗”。因此数据库一旦被破坏仍要用恢复技术把数据库加以恢复。

总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。

恢复的基本原理十分简单。可以用一个词来概括：冗余。这就是说，数据库中任何一部分被破坏或不正确的数据可以根据存储在系统别处的冗余数据来重建。尽管恢复的基本原理很简单，但实现技术的细节却相当复杂，下面略去一些细节，介绍数据库恢复的实现技术。

10.4 恢复的实现技术

恢复机制涉及的两个关键问题是：如何建立冗余数据，以及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（logging）。通常在一个数据库系统中，这两种方法是一起使用的。

10.4.1 数据转储

数据转储是数据库恢复中采用的基本技术。所谓转储即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。这些备用的数据称为**后备副本**（backup）或后援副本。

当数据库遭到破坏后可以将后备副本重新装入，但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。例如，在图 10.1 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，然后重新运行自 T_b～T_f 时刻的所有更新事务，这样就把数据库恢复到故障发生前的一致状态。

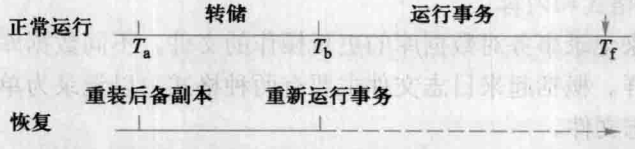

图 10.1 转储和恢复

转储是十分耗费时间和资源的，不能频繁进行。数据库管理员应该根据数据库使用情况确定一个适当的转储周期。

转储可分为静态转储和动态转储。

静态转储是在系统中无运行事务时进行的转储操作。即转储操作开始的时刻数据库处于一致性状态，而转储期间不允许（或不存在）对数据库的任何存取、修改活动。显然，静态转储得到的一定是一个数据一致性的副本。

静态转储简单，但转储必须等待正运行的用户事务结束才能进行。同样，新的事务必须等待转储结束才能执行。显然，这会降低数据库的可用性。

动态转储是指转储期间允许对数据库进行存取或修改。即转储和用户事务可以并发执行。

动态转储可以克服静态转储的缺点，它不用等待正在运行的用户事务结束，也不会影响新事务的运行。但是，转储结束时后援副本上的数据并不能保证正确有效。例如，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。

为此，必须把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**（log file）。这样，后援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。

转储还可以分为海量转储和增量转储两种方式。海量转储是指每次转储全部数据库，增量转储则指每次只转储上一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一般说来会更方便些。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。

数据转储有两种方式，分别可以在两种状态下进行，因此数据转储方法可以分为 4 类：动态海量转储、动态增量转储、静态海量转储和静态增量转储，如表 10.1 所示。

表 10.1 数据转储分类

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

10.4.2 登记日志文件

1. 日志文件的格式和内容

日志文件是用来记录事务对数据库的更新操作的文件。不同数据库系统采用的日志文件格式并不完全一样。概括起来日志文件主要有两种格式：以记录为单位的日志文件和以数据块为单位的日志文件。

对于以记录为单位的日志文件，日志文件中需要登记的内容包括：

- 各个事务的开始（BEGIN TRANSACTION）标记。
- 各个事务的结束（COMMIT 或 ROLLBACK）标记。
- 各个事务的所有更新操作。

这里每个事务的开始标记、每个事务的结束标记和每个更新操作均作为日志文件中的一个日志记录（log record）。

每个日志记录的内容主要包括：

- 事务标识（标明是哪个事务）。
- 操作的类型（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）。

对于以数据块为单位的日志文件，日志记录的内容包括事务标识和被更新的数据块。由于将更新前的整个块和更新后的整个块都放入日志文件中，操作类型和操作对象等信息就不必放入日志记录中了。

2. 日志文件的作用

日志文件在数据库恢复中起着非常重要的作用，可以用来进行事务故障恢复和系统故障恢复，并协助后备副本进行介质故障恢复。具体作用是：

（1）事务故障恢复和系统故障恢复必须用日志文件。

（2）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。

（3）在静态转储方式中也可以建立日志文件，当数据库毁坏后可重新装入后援副本把数据库恢复到转储结束时刻的正确状态，然后利用日志文件把已完成的事务进行重做处理，对故障发生时尚未完成的事务进行撤销处理。这样不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态，如图 10.2 所示。

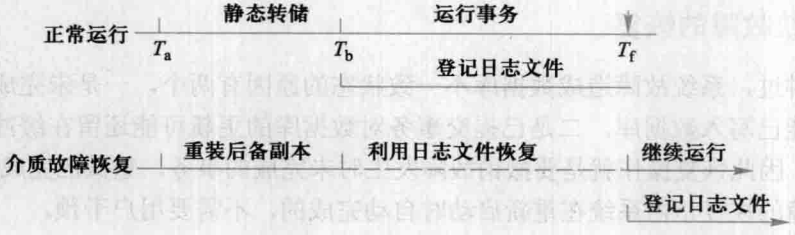

图 10.2 利用日志文件恢复

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。

把对数据的修改写到数据库中和把表示这个修改的日志记录写到日志文件中是两个不同的操作。有可能在这两个操作之间发生故障，即这两个写操作只完成了一个。如果先写了数据库修改，而在运行记录中没有登记这个修改，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。所以为了安全，一定要先写日志文件，即首先把日志记录写到日志文件中，然后写数据库的修改。这就是“先写日志文件”的原则。

10.5 恢复策略

当系统运行过程中发生故障，利用数据库后备副本和日志文件就可以将数据库恢复到故障前的某个一致性状态。不同故障其恢复策略和方法也不一样。

10.5.1 事务故障的恢复

事务故障是指事务在运行至正常终止点前被终止，这时恢复子系统应利用日志文件撤销（UNDO）此事务已对数据库进行的修改。事务故障的恢复是由系统自动完成的，对用户是透明的。系统的恢复步骤是：

（1）反向扫描日志文件（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。这样，如果记录中是插入操作，则相当于做删除操作（因此时“更新前的值”为空）；若记录中是删除操作，则做插入操作；若是修改操作，则相当于用修改前值代替修改后值。

（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。

10.5.2 系统故障的恢复

前面已讲过，系统故障造成数据库不一致状态的原因有两个，一是未完成事务对数据库的更新可能已写入数据库，二是已提交事务对数据库的更新可能还留在缓冲区没来得及写入数据库。因此恢复操作就是要撤销故障发生时未完成的事务，重做已完成的事务。

系统故障的恢复是由系统在重新启动时自动完成的，不需要用户干预。

系统的恢复步骤是：

（1）正向扫描日志文件（即从头扫描日志文件），找出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），将其事务标识记入重做队列（REDO-LIST）。同时找出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），将其事务标识记入撤销队列（UNDO-LIST）。

（2）对撤销队列中的各个事务进行撤销（UNDO）处理。

进行撤销处理的方法是，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）对重做队列中的各个事务进行重做处理。

进行重做处理的方法是：正向扫描日志文件，对每个重做事务重新执行日志文件登记的操作，即将日志记录中“更新后的值”写入数据库。

10.5.3 介质故障的恢复

发生介质故障后，磁盘上的物理数据和日志文件被破坏，这是最严重的一种故障，恢复方法是重装数据库，然后重做已完成的事务。

（1）装入最新的数据库后备副本（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。

对于动态转储的数据库副本，还需同时装入转储开始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。

（2）装入相应的日志文件副本（转储结束时刻的日志文件副本），重做已完成的事务。即首先扫描日志文件，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。

这样就可以将数据库恢复至故障前某一时刻的一致状态了。

介质故障的恢复需要数据库管理员介入，但数据库管理员只需要重装最近转储的数据库副本和有关的各日志文件副本，然后执行系统提供的恢复命令即可，具体的恢复操作仍由数据库管理系统完成。

10.6 具有检查点的恢复技术

利用日志技术进行数据库恢复时，恢复子系统必须搜索日志，确定哪些事务需要重做，哪些事务需要撤销。一般来说，需要检查所有日志记录。这样做有两个问题，一是搜索整个日志将耗费大量的时间，二是很多需要重做处理的事务实际上已经将它们的更新操作结果写到了数据库中，然而恢复子系统又重新执行了这些操作，浪费了大量时间。为了解决这些问题，又发展了具有检查点的恢复技术。这种技术在日志文件中增加一类新的记录——**检查点**（checkpoint）记录，增加一个重新开始文件，并让恢复子系统在登录日志文件期间动态地维护日志。

检查点记录的内容包括：

- 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。

重新开始文件用来记录各个检查点记录在日志文件中的地址。图 10.3 说明了建立检查点 C_i 时对应的日志文件和重新开始文件。

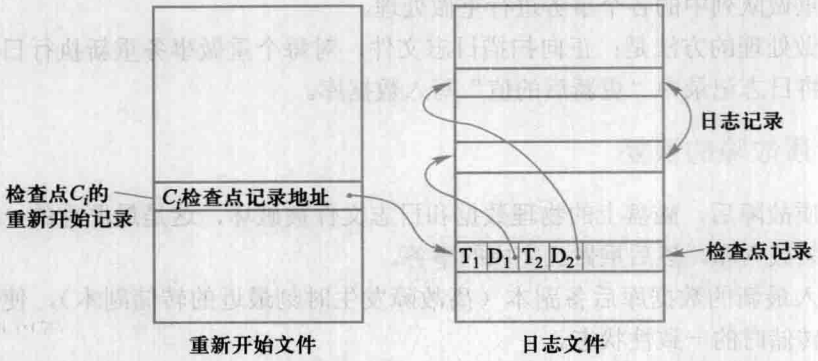

图 10.3 具有检查点的日志文件和重新开始文件

动态维护日志文件的方法是，周期性地执行建立检查点、保存数据库状态的操作。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。

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。检查点可以按照预定的一个时间间隔建立，如每隔一小时建立一个检查点；也可以按照某种规则建立检查点，如日志文件已写满一半建立一个检查点。

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一定都已写入数据库，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。这样，在进行恢复处理时，没有必要对事务 T 执行重做操作。

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，如图 10.4 所示。

T_1：在检查点之前提交。

T_2：在检查点之前开始执行，在检查点之后故障点之前提交。

T_3：在检查点之前开始执行，在故障点时还未完成。

T_4：在检查点之后开始执行，在故障点之前提交。

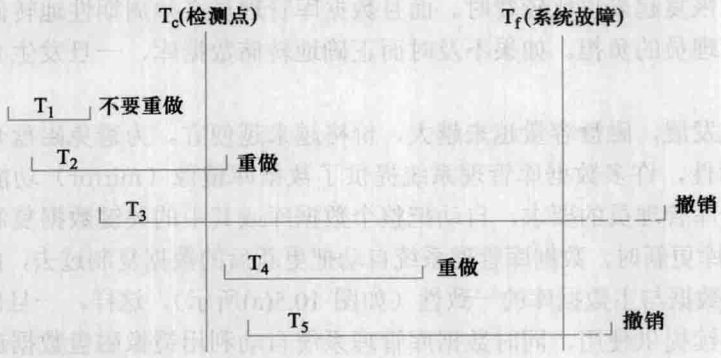

图 10.4 恢复子系统采取的不同策略

T_5：在检查点之后开始执行，在故障点时还未完成。

T_3 和 T_5 在故障发生时还未完成，所以予以撤销；T_2 和 T_4 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，所以要重做；T_1 在检查点之前已提交，所以不必执行重做操作。

系统使用检查点方法进行恢复的步骤是：

（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。

（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。

这里建立两个事务队列：

- UNDO-LIST: 需要执行 UNDO 操作的事务集合；
- REDO-LIST: 需要执行 REDO 操作的事务集合。

把 ACTIVE-LIST 暂时放入 UNDO-LIST 队列，REDO 队列暂为空。

（3）从检查点开始正向扫描日志文件。

① 如有新开始的事务 T_i，把 T_i 暂时放入 UNDO-LIST 队列；

② 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。

10.7 数据库镜像

如前所述，介质故障是对系统影响最为严重的一种故障。系统出现介质故障后，用户

应用全部中断，恢复起来也比较费时。而且数据库管理员必须周期性地转储数据库，这也加重了数据库管理员的负担。如果不及时而正确地转储数据库，一旦发生介质故障，会造成较大的损失。

随着技术的发展，磁盘容量越来越大，价格越来越便宜。为避免磁盘介质出现故障影响数据库的可用性，许多数据库管理系统提供了数据库镜像（mirror）功能用于数据库恢复。即根据数据库管理员的要求，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，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，由数据库管理系统自动保证镜像数据与主数据库的一致性（如图 10.5(a)所示）。这样，一旦出现介质故障，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（如图 10.5(b)所示）。在没有出现故障时，数据库镜像还可以用于并发操作，即当一个用户对数据加排他锁修改数据时，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。

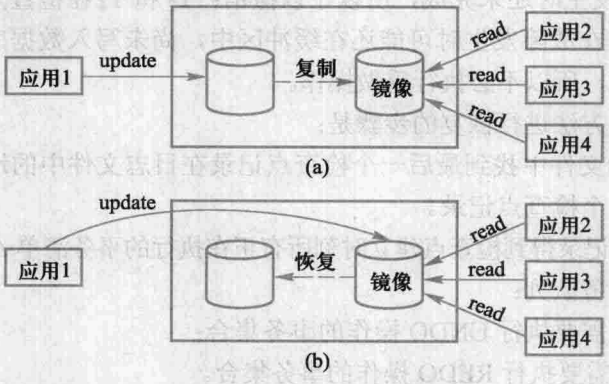

图 10.5 数据库镜像

由于数据库镜像是通过复制数据实现的，频繁地复制数据自然会降低系统运行效率，因此在实际应用中用户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。

10.8 小 结

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，只要数据库管理系统能够保证系统中一切事务的 ACID 特性，即事务的原子性、一致性、隔离性和持续性，也就保证了数据库处于一致状态。为了保证事务的原子性、一致性与持续性，数据库管理系统必须对事务故障、系统故障和介质故障进行恢复。数据转储和登记日志文件是恢复中最经常使用的技术。恢复的基本原理就是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。

事务不仅是恢复的基本单位，也是并发控制的基本单位。为了保证事务的隔离性和一致性，数据库管理系统需要对并发操作进行控制，第 11 章将进一步讲解并发控制。

习 题

1. 试述事务的概念及事务的 4 个特性。恢复技术能保证事务的哪些特性？
2. 为什么事务非正常结束时会影响数据库数据的正确性？请举例说明之。
3. 登记日志文件时为什么必须先写日志文件，后写数据库？
4. 考虑下图所示的日志记录：

序号	日志
1	T_1：开始
2	T_1：写 A，A=10
3	T_2：开始
4	T_2：写 B，B=9
5	T_1：写 C，C=11
6	T_1：提交
7	T_2：写 C，C=13
8	T_3：开始
9	T_3：写 A，A=8
10	T_2：回滚
11	T_3：写 B，B=7
12	T_4：开始
13	T_3：提交
14	T_4：写 C，C=12

（1）如果系统故障发生在 14 之后，说明哪些事务需要重做，哪些事务需要回滚。

（2）如果系统故障发生在 10 之后，说明哪些事务需要重做，哪些事务需要回滚。

（3）如果系统故障发生在 9 之后，说明哪些事务需要重做，哪些事务需要回滚。

（4）如果系统故障发生在 7 之后，说明哪些事务需要重做，哪些事务需要回滚。

5. 考虑题 4 所示的日志记录，假设开始时 A、B、C 的值都是 0：

（1）如果系统故障发生在 14 之后，写出系统恢复后 A、B、C 的值；

（2）如果系统故障发生在 12 之后，写出系统恢复后 A、B、C 的值；

（3）如果系统故障发生在 10 之后，写出系统恢复后 A、B、C 的值；

（4）如果系统故障发生在 9 之后，写出系统恢复后 A、B、C 的值；

（5）如果系统故障发生在 7 之后，写出系统恢复后 A、B、C 的值；

（6）如果系统故障发生在 5 之后，写出系统恢复后 A、B、C 的值。

6. 针对不同的故障，试给出恢复的策略和方法。（即如何进行事务故障的恢复，如何进行系统故障的恢复，以及如何进行介质故障的恢复。）

7. 什么是检查点记录？检查点记录包括哪些内容？

8. 具有检查点的恢复技术有什么优点？试举一个具体例子加以说明。

9. 试述使用检查点方法进行恢复的步骤。

10. 什么是数据库镜像？它有什么用途？

实　验

实验 10　数据库恢复技术

掌握和使用数据库事务管理原理以及事务编程方法，包括显式事务定义、事务提交和回滚。掌握数据库备份原理和方法，能够针对各种备份方法设计备份方案，利用数据库管理系统提供的备份工具实现各种数据库备份策略。掌握数据库恢复原理和方法，能够针对各种备份和恢复方法设计数据库恢复方案，利用数据库管理系统提供的恢复工具实现各种数据库恢复策略。

本章参考文献

[1] DAVIES C T. Recovery Semantics for a DB/DC System. Proceedings of the ACM Annual Conference, 1973.

[2] BJORK L A. Recovery Scenario for a DB/DC System. Proceedings of the ACM Annual Conference, 1973.

（文献［1］和［2］是系统恢复领域中两篇最早的论文。）

[3] HAERDER T, REUTER A. Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys, 1983(15):4.

[4] JIM G, et al. The Recovery Manager of the System R Data Manager. ACM Computing Surveys, 1981(13):2.

（文献［4］概述了 IBM 的数据库管理系统（实验系统）System R 的恢复技术。）

[5] CRUS R A. Data Recovery in IBM Database 2. IBM Sys, 1984(23):2.

（文献［5］介绍了 IBM DB2 的恢复技术。）

[6] LORIE R A. Physical Integrity in a Large Segmented Database. ACM Transactions on Database Systems, 1977(2):1.

（文献［6］研究了 System R 的影子页面技术。）

[7] CHANDY K M, BROWNE J C, DISSLEY C W, et al. Analytic Models for Rollback and Recovery Strategies in Database Systems. IEEE Transactions on Software Engineering. 1975, SE-1:1.

（文献［7］提出了数据库系统恢复和回滚的分析模型。）

[8] REUTER A. A Fast Transaction-Oriented Logging Scheme for UNDO Recovery. IEEE Transactions on Software Engineering. 1980, SE-6:4.

（文献［8］给出了一个快速处理 UNDO 操作的日志模式。）

[9] LILIEN L, BHARGAVA B. Database Integrity Block Construct: Concept and Design Issues. IEEE Transactions on Software Engineering. 1985, SE-11:9.

（文献［9］介绍了完整性、并发控制和系统恢复相结合的问题。）

[10] VERHOFSTAD J S M. Recovery Techniques for Database Systems. ACM Computing Surveys. 1978, 10:2.

（文献［10］系统地综述了数据库管理系统的恢复技术。）

[11] BERNSTEIN A, HADZILACOS V, GOODMAN W. Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.

（文献［11］包含了有关 DBMS 恢复的全面介绍。）

[12] MOHAN C, HADERLE D, LINDSAY R, et al. ARIES: A Transaction Recovery Method Supporting Fine Granularity Locking and Partial Rollback Using Write-Ahead Logging. ACM Transactions on Database Systems, 1991.

（文献［12］讨论了 ARIES 系统中的事务恢复技术。）