第 4 章　数据库安全及维护

本章学习目标

数据库系统中由于大量数据集中存放，而且为许多用户直接共享，是宝贵的信息资源，所以安全性问题更为重要。系统安全保护措施是否有效是数据库系统的主要性能指标之一。本章将介绍以下几部分内容:

- 数据库安全性问题的基本概念和保证数据安全性的基本措施
- 数据库并发情况下的数据安全及处理方法
- 数据发生故障情况下的恢复技术

4.1　数据库安全性

4.1.1　基本概念

数据库的安全性指保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。

安全性问题不是数据库系统所独有的，计算机系统都有这个问题，是因为在数据库系统中，大量数据集中存放，并为许多用户直接共享，是宝贵的信息资源，从而使安全性问题更为突出，系统安全保护措施是否有效是数据库系统的主要性能指标之一。

安全性问题和保密问题是密切相关的。前者主要涉及数据的存取控制、修改和传播的技术手段；而后者在很大程度上是法律、政策、伦理、道德等问题。这一节主要讨论安全性的一般概念和方法，然后介绍一些数据库系统的安全性措施。

4.1.2　安全措施的设置模型

在计算机系统中，安全措施通常是一级一级层层设置的。例如可以有如图 4-1 所示的计算机系统安全模型。

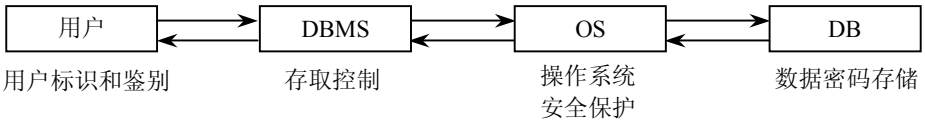

图 4-1　计算机系统安全模型

在如图 4-1 所示的安全模型中，用户在进入计算机系统时，首先根据用户输入的用户标识进行身份验证，只有合法的用户才准许登录计算机系统。进入计算机系统后，DBMS 对用户进行存取控制，只允许用户执行合法操作，操作系统同时允许用户进行安全操作。数据最终可

以通过加入存取密码的方式存储到数据库中。

下面讨论最常见的用户标识与鉴别、存取控制等安全措施。

1. 用户标识和鉴别

首先，系统提供一定的方式让用户标识自己的名字或身份。系统进行核实，通过鉴别后才提供机器使用权。常用的方法有：

- 用一个用户名或者用户标识号来标明用户身份，系统鉴别此用户是否为合法用户。若是，则可以进入下一步的核实；若不是，则不能使用计算机。
- 口令（Password）：为了进一步核实用户，系统常常要求用户输入口令。为保密起见，用户在终端上输入的口令通常显示为其他符号（Windows 环境下通常显示为*），系统核对口令以鉴别用户身份。该方法简单易行，但用户名、口令容易被人窃取，因此还可用更复杂的方法。
- 系统提供一个随机数，用户根据预先约定好的某一过程或函数进行计算，系统根据用户计算结果是否正确进一步鉴别用户身份。

2. 存取控制

当用户通过了用户标识和鉴别后，要根据预先定义好的用户权限进行存取控制，保证用户只能存取自己有权存取的数据。这里的用户权限是指不同的用户对于不同的数据对象允许执行的操作权限，它由两部分组成：一是数据对象，二是操作类型，定义用户的存取权限就是要设置该用户可以在哪些数据对象上进行哪些类型的操作。表 4-1 列出了关系系统中的数据对象内容和操作类型。

表 4-1　关系系统中的存取权限

数据对象	操作类型
模式、外模式、内模式	建立、修改、使用（检索）
表或记录、字段	查找、插入、修改、删除

表中的数据对象由两类组成：一类是数据本身，如关系数据库中的表、字段，非关系数据库中的记录、字段（亦称为数据项）；另一类是外模式、模式、内模式。在非关系系统中，外模式、模式、内模式的建立、修改均由数据库管理员（DBA）负责，一般用户无权执行这些操作，因此存取控制的数据对象仅限于数据本身。在关系系统中 DBA 可以把建立、修改基本表的权力授予用户，用户获得此权力后可以建立基本表、索引、视图。所以，关系系统中存取控制的数据对象不仅有数据，而且有模式、外模式、内模式等数据字典中的内容。

下面讨论数据库中存取控制的一般方法和技术。

定义用户存取权限称为授权（Authorization）。这些定义经过编译后存储在数据字典中。每当用户发出存取数据库的操作请求后，DBMS 查找数据字典，根据数据字典中的用户权限进行合法权检查（Authorization Check）。若用户的操作请求超出了定义的权限，系统拒绝执行此操作，授权编译程序和合法权检查机制一起组成了安全性子系统。例如，在第 4 章中讲到的 SQL 语句中的授权、回收权限语句就是这种工作原理。

4.2　数据库的完整性

4.2.1　基本概念

1. 数据库的完整性

数据库的完整性指数据的正确性和相容性，DBMS 必须提供一种功能来保证数据库中数据的完整性，这种功能亦称为完整性检查，即系统用一定的机制来检查数据库中的数据是否满足规定的条件，这种条件在数据库中称为完整性约束条件，这些完整性约束条件将作为模式的一部分存入数据库中。

2. 数据的完整性和安全性的区别

数据的完整性和安全性是两个不同的概念，数据的完整性是为了防止数据库中存在不符合语义的数据，防止错误信息的输入和输出，即所谓垃圾进垃圾出（Garbage In Garbage Out）所造成的无效操作和错误结果；数据的安全性是保护数据库以防止恶意的破坏和非法的存取，当然，完整性和安全性是密切相关的，如果从系统实现的方法来看，某一种机制常常既可用于安全性保护也可用于完整性保证。

4.2.2　完整性约束

完整性约束条件可以进行以下分类：

1. 值的约束和结构的约束

值的约束是对数据值的限制，结构的约束是指对数据之间联系的限制

（1）关于对数据值的约束。这类约束条件是指对数据取值类型、范围、精度等的规定，例如：

- 对某个属性和属性组合规定某个值集，教师年龄是大于或等于 16，小于或等于 65 的整数或实数，若为实数，可规定精度为小数点后一位数字。又如，规定年份是四位整数，月份是一至十二的整数。规定零件颜色是红、绿、黄、黑、白五种。
- 规定某属性值的类型和格式。如规定学号第一个字符必须是字母，后面是 7 位数字，规定学生名字是字符串。
- 规定某属性的值的集合必须满足某种统计条件。如任何职工的工资不得超过此部门平均工资的三倍，任何职工的奖金不得超过此部门平均工资的 30%。

（2）关于数据之间联系的约束。数据库中同一关系的不同属性之间可以有一定的联系，从而应满足一定的约束条件，同时，由于数据库中数据是结构化的，不同的关系之间也可以有联系，因而不同关系的属性之间也可满足一定的约束条件，例如：数据之间的相互关联最常见的是通过值的相等与否来体现。又如，任何一个关系必存在一个或多个候选键。任何一个键值唯一地确定关系的一个元组，因此候选键的值在关系中必须是唯一的，主键值也必须非空（关系模型的实体完整性）。

一个关系中某属性的值集是同一关系或另一关系中某属性值集的子集。如在前面提到的几个基本表中，选课表中的课程号 c_number 必须是课程表 c_number 的子集。这个例子是关系模型的参照完整性，即一个关系的外键的值集一定是相应的另一个关系上主键属性值集的子集。

2. 静态约束和动态约束

静态约束是指对数据库每一确定状态的数据所应满足的约束条件。以上所讲的约束都属于静态约束。

动态约束是指数据库从一种状态转变为另一种状态时新、旧值之间所应满足的约束条件。例如，当更新教师工资时要求新工资值不低于旧工资值。又如，当学生学费大于等于 3500 元时新学费等于旧学费，否则新学费大于或等于旧学费，这条约束体现了这样的语义：调整收费范围仅限于学费低于 3500 元的学生。

3. 立即执行约束和延迟执行约束

立即执行约束是指在执行用户事务时，对事务中某一更新语句执行完后马上对此数据所应满足的约束条件进行完整性检查。

延迟执行约束是指在整个事务执行结束后才对此约束条件进行完整性检查，结果正确才能提交。

4. 完整性的实现

完整性的实现应包括以下两个方面：

- 系统要提供定义完整性约束条件的功能。
- 系统提供检查完整性约束条件的方法。

对于数据值的那类完整性约束条件通常在模式中定义。例如在模式中定义属性名、类型、长度、码属性名并标明其值是唯一的、非空的等，另外的那些约束条件就要用专门的方式加以定义，如编写程序实现约束。

4.3 并发控制

4.3.1 基本概念

数据库是一个共享资源，可以由多个用户使用。这些用户程序可以一个一个地串行执行。每个时刻只有一个用户程序运行，执行对数据库的存取。其他用户程序必须等到这个用户程序结束以后才能对数据库存取。如果一个程序执行时，用户进行大量数据的输入/输出工作，则数据库系统的大部分时间将处于休闲状态，为了充分利用数据库资源，应该允许多个用户程序并行地存取数据库，这样就会产生多个用户程序并发地存取同一数据的情况。若对并发操作不加控制就会存储和读取不正确的数据，破坏数据库的完整性（一致性）。

1. 事务

事务（Transaction）是并发控制的单位，是一个操作序列。这些操作要么都做，要么都不做，是一个不可分割的工作单位。

事务通常以 BEGIN TRANSACTION 开始，以 COMMIT 或 ROLLBACK 操作结束。COMMIT 即提交，提交事务中所有的操作，事务正常结束；ROLLBACK 即撤消已作的所有操作，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。

事务和程序是两个概念，一般地讲，程序可包括多个事务。由于事务是并发控制的基本单位，所以下面的讨论均以事务为对象。

2. 数据一致性级别的概念

首先分析火车票订票系统中的一个活动序列（如表 4-2 所示）：

①甲售票点读出某车次的车票余额 A，设 A＝20。

②乙售票点读出同一车次的车票余额 A，也为 20。

③甲售票点卖出一张车票，修改余额 A←A-1。所以 A 为 19，把 A 写回数据库。同时乙售票点也卖出一张车票，修改余额 A←A-1。所以 A 为 19，把 A 写回数据库。结果，卖出两张车票而余额只减少 1。

表 4-2　数据的不一致性

丢失修改		不能重复读取		读“脏”数据	
T_1	T_2	T_1	T_2	T_1	T_2
①读 A=20		①读 A=50 读 B=100 求和=150		①读 C=100 C=C*2 写回 C	
②	读 A=20	②	读 B=100 B=B*2 写回 B	②	读 C=200
③A=A-1 写回 A=19		③		④ROLLBACK C 恢复为 100	
④	A=A-1 写回 A=19	④读 A=50 读 B=200 求和=250 （验算不对）		④	

数据的不一致性：因多个事务对同一数据的交叉修改（并发操作）而引起的数据不正确或数据修改丢失就称为数据的不一致性。

上例中，因为在并发操作情况下，对甲、乙两个事务的操作序列的调度是随机的，若按上面的调度序列执行，甲事务的修改就被丢失。这是由于第③步中乙事务修改 A 并写回后破坏了甲事务的修改。

仔细分析并发操作，可能会产生以下几种不一致性：

- 丢失修改：两个事务 T_1 和 T_2 读入同一数据并修改，T_2 提交的结果破坏了 T_1 提交的结果，T_1 的修改被丢失。上面预订车票例子就属于此类。
- 不能重复读取：事务 T_1 读取某一数据，事务 T_2 读取并修改了同一数据，T_1 为了对读取值进行校对再读此数据，得到了不同的结果。例如 T_1 读取 B=100，T_2 读取 B 并把 B 改为 200，T_1 再读 B 得 200，与第一次读取值不一致。
- 读“脏”数据：事务 T_1 修改某一数据，事务 T_2 读取同一数据，而 T_1 由于某种原因被撤消，则 T_2 读到的数据就为“脏”数据，即不正确的数据，例如 T_1 把 C 由 100 改为 200，T_1 读到 C 为 200，而事务 T_1 由于被撤消，其修改宣告无效，C 恢复为原

值 100，而 T_2 却读到了 C 为 200，与数据库内容不一致。

3. 并发控制

并发控制就是要用正确的方式调度并发操作，避免造成数据的不一致性，使一个用户事务的执行不受其他事务的干扰。

另一方面，对数据库的应用有时允许某些不一致性。例如，有些统计工作涉及数据量很大，读到一些脏数据对统计精度没什么影响，这时可以降低对一致性的要求以减少系统开销。

并发控制的主要方法是采用封锁机制。例如在火车订票例子中，若甲事务要修改 A 时，在读出 A 前先封锁 A。这时其他事务就不能读取和修改，直到甲修改并写回 A 后解除了对 A 的封锁为止，这样，就不会丢失甲的修改。

4.3.2 封锁

封锁：事务对数据库操作之前，先对数据加锁以便获得这个数据对象的一定控制，使得其他事务不能更新此数据，直到该事务解锁为止。

1. 封锁的类型

- 共享性封锁（共享锁，或称 S 锁），也称读锁（RLOCK）：若事务 T 对数据对象 A 加上 S 锁，则事务 T 可以读取 A 但不能修改 A，其他事务只能对 A 加 S 锁，而不能加 X 锁，直到 T 释放 A 上的 S 锁。这就保证了其他事务可以读 A，但在 T 释放 A 上的锁之前不能修改 A。
- 排他性封锁（排他锁，或称 X 锁），也称写锁（WLOCK）：若事务 T 对数据对象 A 加上 X 锁，则只允许 T 读取和修改 A，其他任何事务都不能再对 A 加任何类型的锁，直到 T 释放 A 上的锁。这就保证了其他事务在 T 释放 A 上的锁之前不能再读取和修改 A。

2. 封锁类型的控制方式

封锁类型决定控制方式，用相容矩阵表示控制方式，如表 4-3 所示。在表 4-3 中，最左列表示事务 T_1 已经获得的数据对象上的锁的类型，其中“－”表示没有加锁；最上面一行表示另一事务 T_2 对同一数据对象发出的封锁请求。T_2 的封锁请求能否被满足用矩阵中的 Y 和 N 表示，其中 Y 表示事务 T_2 封锁请求与 T_1 已经获得的锁相容，封锁请求可以满足。N 表示事务 T_2 封锁请求与 T_1 已经获得的锁冲突，封锁请求被拒绝。

表 4-3 封锁类型的相容性

T_1 \ T_2	X 锁	S 锁	－
X 锁	N	N	Y
S 锁	N	Y	Y
－	Y	Y	Y

3. 用封锁机制解决购买火车票问题

用封锁机制解决购买火车票问题，如表 4-4 所示。

表 4-4　封锁机制解决火车票问题

数据修改后不丢失		能重复读取		不再读“脏”数据	
T_1	T_2	T_1	T_2	T_1	T_2
①X 锁→A 读 A=20		①S 锁→A S 锁→B 读 A、B 求和=150		①X 锁→C 读 C=100 C=C*2 写回 C	
②	请求 X 锁→A	②	请求 X 锁→B	②	请求 S 锁→C
③A=A-1 写回 A=19 提交 解锁→A	等待	③读 A、B 求和=150 解锁→A 解锁→A	等待	③回滚 C 恢复为 100	等待
④	X 锁→A 成功 读 A=19	④	X 锁→B 成功 读 B=100	④	S 锁→C 成功 读 C=100
⑤	A=A-1 写回 A=18 提交 解锁→A	⑤	B=B*2 写回 B=200 提交 解锁→B	⑤	

4. 活锁与死锁

（1）活锁：某一事务的请求可能永远得不到，该事务一直处于等待状态。

解决方法：按照请求封锁的先后顺序排队，先来先服务策略。

（2）死锁：两个事务处于相互等待状态，永远不能结束。

解决方法：

- 将所有数据一次性加锁——降低了并发度。
- 预先规定一个封锁顺序。
- 诊断法：检测是否有死锁发生，如有则设法解除。

4.3.3　并发调度的串行操作和并行操作

1. 对多个事务处理的两种方法

串行操作：一个个地处理，一组事务的任意串行操作都可以保证数据的一致性。

并行操作：利用分时的方法同时处理多个事务。

2. 调度

把并发操作的全部事务按某一顺序运行。

3. 串行调度

若多个事务按照某一次序串行地执行，则称事务的调度是串行的。

4. 并行调度

若多个事务同时交叉地并行执行，则称事务的调度是并行的。

4.4 数据恢复

尽管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，保证并行事务的正确执行，但是计算机系统中硬件的故障、软件的错误、操作员的失误以及故意的破坏仍是不可避免的，这些故障轻则造成正在运行的事务非正常地中断，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（亦称为完整状态或一致状态）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（如 MS SQL Server、DB2）。故障恢复是否考虑周到和行之有效的，是数据库系统性能的一个重要指标。

4.4.1 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，有的是非预期的、不能由事务程序处理的。

例如，发票管理软件中，开完一张票据后，数据存盘程序如下（PowerScript 语言）：

```
IF dw_1.Update() > 0 THEN
    Commit;
ELSE
    MessageBox("发票管理","对不起，无法对数据保存!")
    Rollback;
END IF
```

这个例子说明事务是一个“完整的”工作单位，它所包括的一组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。由于新输入的数据是一条一条记录存入到数据库中的，在这段程序中若数据存储到数据库一半时发生错误，dw_1.Update()函数运行返回值会小于 0，这时应用程序可以发现并让事务回滚，撤消刚才存储到数据库中的所有记录，恢复数据库到初始正确状态。

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、并行事务发生死锁而被选中撤消该事务等，以后事务故障仅指这一类故障。

事务故障意味着事务没有到达预期的终点（COMMIT 或者显式的 ROLLBACK），因此，数据库可能处于不正确状态，系统就要强行回滚此事务，即撤消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。

（2）系统范围内的故障。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 CPU 故障、操作系统故障、突然断电等。这类故障影响正在运行的所有事务，但不破坏数据库。这时主存内容，尤其是数据库缓冲区（在内存）中的内容都被丢失，使得运行事务都非正常终止，从而造成数据库可能处于不正确的状态，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滚回，把数据库恢复到正确的状态。

（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，如磁盘的磁头碰撞、瞬时的强磁场干扰。这类故障将破坏数据库或部分数

据库，并影响正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。

病毒的种类很多，目前世界上发现的病毒及其变种数量已近 5 万种。不同病毒有不同的特征，小的病毒只有 20 条指令，不到 50 字节。大的病毒像一个操作系统，由上万条指令组成。目前，绝大多数病毒是在 IBM PC 及其兼容机之间传播的。

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，机器在感染后两、三年才开始发病；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，它们只在数据库中或其他数据文件中将小数点向左或向右移一移，增加或删除一两个“0”。

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查杀病毒软件、安全“疫苗”。但是，至今还没有一种使得计算机能够“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，主要有两类：一是数据库本身被破坏，二是数据库没有破坏，但数据可能不正确，这是因为事务的运行被终止造成的。

数据恢复的基本原理十分简单，采用的主要技术手段是冗余，这就是说，数据库中任何一部分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。下面将略去许多细节，介绍目前数据库系统中最常用的两种方法：转储和登记日志文件。

4.4.2　转储和恢复

1. 什么是转储和恢复

转储：数据库恢复中采用的基本技术。所谓转储即 DBA（数据库管理员）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。

恢复：当数据库遭到破坏后就可以利用后备副本把数据库恢复，这时数据库只能恢复到转储时的状态，从那以后的所有更新事务必须重新运行才能恢复到故障时的状态。

转储是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一个适当的转储周期。

2. 转储的分类

静态转储：指转储期间不允许（或不存在）对数据库进行任何存取、修改活动。静态转储简单，但转储必须等待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：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，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，在转储期间的某时刻 T_1 系统把数据 A=100 转储到了磁带上，而在时刻 T_2，某一事务对 A 进行了修改，使 A=200。转储结束，后援副本上的 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。

海量转储：海量转储是指每次转储全部数据库。

增量转储：增量转储则指每次只转储上次转储后更新过的数据。如果数据库很大，事务处理又十分频繁，则增量转储方式是很有效的。

4.4.3 日志文件

日志文件：用来记录对数据库每一次更新活动的文件。

在动态转储方式中必须建立日志文件，后援副本和日志文件综合起来才能有效地恢复数据库。在静态转储方式中，也可以建立日志文件，当数据库被破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消处理。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。

下面介绍如何登记日志文件以及发生故障后如何利用日志文件恢复事务。

1. 登记日志文件（logging）

事务在运行过程中，系统把事务开始、事务结束（包括 COMMIT 和 ROLLBACK），以及对数据库的插入、删除、修改等每一个操作作为一个登记记录（log 记录）存放到日志文件中。每个记录包括的主要内容有：执行操作的事务标识、操作类型、更新前数据的旧值（对插入操作而言此项为空值）、更新后的新值（对删除操作而言此项为空值）。

登记的次序严格按并行事务操作执行的时间次序，同时遵循“先写日志文件”的原则。我们知道写一个修改到数据库中和写一个表示这个修改的 log 记录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下这个修改，则以后就无法恢复这个修改。因此为了安全应该先写日志文件，即首先把 log 记录写到日志文件上，然后写数据库的修改。这就是“先写日志文件”的原则。

2. 用日志文件恢复事务

利用日志文件恢复事务的过程分为以下两步：

（1）从头扫描日志文件，找出哪些事务在故障发生时已经结束（这些事务有 BEGIN TRANSACTION 和 COMMIT 记录），哪些事务尚未结束（这些事务只有 BEGIN TRANSACTION 记录，无 COMMIT 记录）。

（2）对尚未结束的事务进行撤消（也称为 UNDO）处理。进行 UNDO 处理的方法是，反向扫描日志文件，对每个 UNDO 事务的更新操作执行反操作。即对已经插入的新记录执行删除操作，对已删除的记录重新插入，对修改的数据恢复旧值（用旧值代替新值）。

对已经结束的事务进行重做（REDO）处理。进行 REDO 处理的方法是，正向扫描日志文件，重新执行登记的操作。

对于非正常结束的事务显然应该进行撤消处理，以消除可能对数据库造成的不一致性。对于正常结束的事务进行重做处理也是需要的。这是因为虽然事务已发出 COMMIT 操作，但更新操作有可能只写到了数据库缓冲区（在内存），还没有来得及物理地写到数据库中（外存）便发生了系统故障，数据库缓冲区的内容被破坏，这种情况仍可能造成数据库的不一致性。由

于日志文件上事务的更新活动已完整地登记下来，因此可以重做这些操作而不必重新运行事务程序。

4.4.4　用转储和日志文件恢复数据库

利用转储和日志文件可以有效地恢复数据库，当数据库本身被破坏时（如硬故障和病毒破坏），可重装转储的后备副本，然后运行日志文件，执行事务恢复，这样就可以重建数据库。当数据库本身没被破坏，但内容已经不可靠时（如发生事务故障和系统故障）可利用日志文件恢复事务，从而使数据库回到某一正确状态，这时不必重装后备副本。前一种情况需要 DBA 执行恢复过程，后一种情况一般无需 DBA 介入而由系统自动执行。

本章小结

数据的安全性是数据库的一个重要指标，它是保护数据以防止不合法的使用所造成数据泄漏、更改和破坏的有效手段。保证数据安全性的主要措施有用户标识与鉴别、存取控制等安全措施。

数据库的完整性控制包括完整性约束的定义及其对完整性约束的检查和处理。当对数据库进行更新操作时，系统会检查用户的操作是否违反了完整性约束，若违反了完整性约束，就采取一定的措施来保证数据的完整性。

数据的安全性和完整性是两个不同的概念，学习时应注意加以区别。

数据库并发控制的最小单位是事务，也是数据库中一个重要概念。应掌握事务的定义、特性和调度方式，以及事务并发执行所带来的三个问题：丢失修改、不能重复读取、读“脏”数据。解决这些问题的方法是封锁。封锁分为排他锁和共享锁，在封锁的过程中可能发生活锁和死锁现象。对这些基本概念应加深理解。

数据库系统在运行过程中会遇到各种故障，主要包括事务内部故障、系统范围内的故障、介质故障和计算机病毒，可采取的主要恢复技术有转储和登记日志文件。

习题四

1．叙述实现数据库安全性控制的常用方法和技术。
2．举例说明数据库的安全控制方法。
3．什么是数据库的完整性？
4．说明数据的完整性与安全性的不同。
5．简述值的约束和结构的约束的概念，并举例说明。
6．简述动态约束和静态约束的概念，并举例说明。
7．简述事务的定义、特性和调度方式。
8．简述排他锁和共享锁的概念。