第 9 章 数据存储和查询处理与优化

1．试述数据库中文件的记录组织方式。

答：数据库中文件的记录组织方式有定长和变长两种。定长记录是最常见的，最容易直接访问。定长记录的关键是记录的大小。如果记录太小，小于记录存储的字符数，那么多出的字符就要被截掉;如果记录太大，大于要存储的字符数，就会有空间的浪费。 变长记录不会有剩余空间和截掉记录，所以克服了定长记录的两个缺点。但因为记录的位置很难计算，所以直接读取很困难。连续访问的文件或通过目录查找的文件经常使用变长记录格式。

2．数据库中常用的文件组织方式是什么?

答：数据库中常见的文件组织方式有：顺序存储、直接存储和索引。顺序存储指将各条记录顺序地存放在外存的连续区内，记录的物理顺序和逻辑顺序是完全一致的。 它适用于所有的文件媒体。 直接存储方式中记录的逻辑顺序与物理顺序不一定相同，但记录的键值直接指明该记录的地址，所以只要知道了记录键值，就能查找该记录的物理位置。索引方式又分为顺序索引和非顺序索引。顺序索引其记录的物理顺序和逻辑顺序一致，记录按记录键的顺序存放，并带有索引。非顺序索引也带索引表，但文件记录的物理顺序和逻辑顺序不一致，索引表中存有已排序的记录键号及该键号的记录地址。

3．什么叫索引? 索引的组织方式主要有哪几类?

答：索引为一种数据组织结构，他由一系列存储在磁盘上的索引项组成。一个索引项是由两列构成的一张表，对应于索引中的一行。第一列是索引键，由行中某些列(或某一列)中的值串接而成;第二列是行指针，指向行所在的磁盘位置。

索引的组织方式主要分为两类:一类是顺序索引，就是根据值的顺序排列(文件里面的值，也就是为其建索引的字段值，顺序放在索引文件里面);另一类是散列索引，将值平均分配到若干散列桶中，通过散列函数定位一个值所属的散列桶。 顺序索引又可以分为单级索引和多级索引。单级索引就是把所有的索引字段及对应的文件位置按顺序一个个地排列出来。多级索引实际上就是在单级索引的基础上再加索引，也就是指向索引的索引，二级索引上面还可以再加三级索引，可以不停地加，加到最后最上层只剩下一个节点(根节点)，就成了一个树状结构。

4．什么情况下使用稠密索引优于稀疏索引?

答:在顺序文件中,如果每个键都有一个索引键值与之相对应,则这样的索引称为稠密索引。反之， 如果只在索引文件中为每个数据块的第一条记录建立索引键值，则称为稀疏索引。稠密索引文件支持按给定键值查找相应记录的查询。 稀疏索引相较于稠密索引节省了存储空间，但查找给定值的记录需要更多的时间。

5．在 B+树中叶节点与非叶节点是否采用同一种索引技术?

答：B+树索引是将一个或几个属性列的索引键值按照一种次序存放起来，以便达到快速进行数据查找的目的。 叶节点采用稠密索引技术存储索引键值；非叶节点则稀疏索引技术存储指引搜索方向的数据项，相当于是叶子节点的索引。

6．什么叫散列索引?

答：散列索引通过散列函数进行定位。散列文件组织将数据的某个键作为参数带入散列函数 h，计算出一个介于 0~B-1 的整数(B 为桶的数目)，之后将记录连接到桶号为 h(K)的桶中进行存储。

7．简述在 B+树中查找索引项值为 *K* 的过程。

答：首先在索引中查找键值小于或等于“K”的最大键值，然后根据指针找到相应的数据块，之后，搜索这个数据块找到键值为“K”的记录。

8．比较聚集索引和非聚集索引。

答：根据索引键值的顺序与数据表的物理顺序是否相同，可以把索引分成两种类型。若索引键值的逻辑顺序与索引所服务的表中相应行的物理顺序相同，则该索引称为聚集索引，反之为非聚集索引。聚集索引的索引值直接指向数据表对应元组，而非聚集索引的索引值仍会指向下一个索引数据块，并不直接指向元组，所以非聚集索引可以因为不同的键值排序而拥有多个不同的索引。聚集索引因为与表的元组物理顺序一一对应，所以只有一种排序，即一张数据表只有一个聚集索引。聚集索引对于那些经常要搜索范围值的列或查找特定的行特别有效。非聚集索引则适合用于频繁更新列。

9．说明在关系数据库系统中进行查询优化的必要性。

答：每个查询都有许多可供选择的执行策略和操作算法，查询优化就是选择一个高效执行的查询处理策略的过程。查询优化有多种方法，一般可分为代数优化和物理优化。代数优化是指按照一定的规则，改变关系代数表达式中操作的次序和组合，使查询执行更高效。物理优化是指存取路径和底层操作算法的选择。选择的依据可以是基于规则的，也可以是基于代价的，还可以是基于语义的。RDBMS 中的查询优化器综合运用这些优化技术，以获得最好的查询优化效果。

10．　简要说明 RDBMS 查询优化的一般准则。

答：RDBMS 查询优化的一般准则包括：（1）提早执行选择运算。（2）合并笛卡儿积与其后的选择运算为连接运算。(3)将投影运算和选择运算同时进行，以避免重复扫描关系。(4)　将投影运算和其前后的二目运算结合起来，以避免为去掉某些字段再扫描一遍关系。(5)在执行连接前对关系适当地预处理．就能快速地找到要连接的元组。(5)　存储公共子表达式。如果一个表达式中多次出现某一个子表达式，那么应该把该子表达式计算的结果预先计算和保存起来，以便以后使用，减少重复计算的次数。

第 10 章　事务与并发控制

1．试述事务的概念及事务的 4 个特性。

答：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。为了保证数据完整性，要求数据库系统维护事务的 ACID 特性，具体为：（1）原子性（Atomicity）：事务的所有操作在数据库中要么全部正确反映，要么全部不反映。（2）一致性（Consistency）：当事务单独运行时（没有其他事务的并发执行），应保持数据库的一致性。（3）隔离性（Isolation）：当多个事务并发执行时，一个事务的执行不能被其他事务干扰。（4）　持久性（Durability）：一个事务一旦提交，它对数据库中数据的改变应该是永久性的，即使系统可能出现故障。

2．在数据库中为什么要并发控制？并发操作可能产生哪几类数据不一致？如何避免并发引起的问题？

答：多个事务并行执行可能引起许多数据一致性的问题，需要事务管理器进行特殊处理以保证数据的一致性。事务在并发执行过程中如果不加控制，则可能会产生：读脏数据、不可重复读以及丢失更新的情况。为避免上述情况的发生，需要确保事务在并行执行过程中其执行结果的可串行性，为此 DBMS 提供基于锁的并发控制协议，两段锁协议及其增强协议：严格两段锁协议和强两段锁协议，通过对加锁和解锁操作的要求以及对持锁时间的控制，从不同程度上规避并发操作可能产生的问题。

3．什么是封锁？简要介绍基本的封锁类型。

答：封锁是指事务在对数据库进行读、写操作之前，必须先得到对操作对象的控制权力。即先要对将执行读、写操作的数据库对象申请锁，在获得该数据库对象的控制权力后，才能进行相应地读、写操作。根据申请到的锁的不同类型，事务可以对数据对象执行不同的操作。对数据项加锁的方式有多种，最基本的有两种：（1）共享锁（S 锁）：如果事务 Ti 申请到数据项 Q 的共享锁，则 Ti 可以读数据项 Q，但不能写 Q。（2）排它锁（X 锁）：如果事务 Ti 申请到数据项 Q 的排它锁，则 Ti 可以读数据项 Q，也可以写 Q。每个事务都要根据自己将要执行的操作类型向锁管理器申请适当的锁。只有在锁管理器授予所需锁后，事务才能进行相关操作。

4．什么是两段封锁协议？严格两段锁协议？强两段锁协议？

答：两段锁协议是指所有事务分两个阶段提出加锁和解锁申请。

（1）增长阶段（Growing Phase）：在对任何数据进行读、写操作之前，首先申请并获得该数据的封锁。

（2）收缩阶段（Shrinking Phase）：在释放一个锁后，事务不再申请和获得其他的任何锁。

严格两段锁协议除要求满足两段锁协议规定外，还要求事务的排它锁必须在事务提交之后释放。

强两段锁协议除要求满足两段锁协议规定外，还要求事务的所有锁都必须在事务提交之后释放。

5．什么叫活锁？如何防止活锁？

答：事务因一直得不到数据项的封锁而处于长久的等待状态就称为活锁。解决活锁的最简单的方法就是采用先来先服务的策略。

6．什么叫死锁？如何处理死锁？

答：在事务并行执行过程中，假如事务 T1 已获得数据项 A 的封锁，事务 T2 已获得数据项 B 的封锁。事务 T1 申请数据项 B 的封锁，且申请的封锁与事务 T2 已经获得的封锁是不相容的，于是事务 T1 被挂起等待。事务 T2 随后申请数据项 A 的封锁且申请的封锁也与事务 T1 已经拥有的封锁冲突，于是事务 T2 等待。这样就出现了事务 T1 等待事务 T2 释放锁，而事务 T2 也在等待事务 T1 释放锁的情况。两个事务都不可能运行结束，出现死锁。

针对死锁，DBMS 有两种处理方式：一种是进行死锁的预防，通过顺序封锁法、一次封锁法、时间戳等方法不让并发执行的事务出现死锁的状况；另一种是允许死锁的发生，在死锁出现后选取部分事务执行撤销操作，解决死锁。

7．简述串行调度和可串行化调度的差别。

答：若不同事务的活动在调度中是一个接一个执行的，没有交叉运行，就称这样的调度为串行调度。当多个事务交叉调度的结果与某一个串行调度的结果相同时，就说该调度是可串行化的。两者的差别就是是否事务的执行顺序是否是串行的。

8．设 T_1、T_2、T_3是如下的 3 个事务:

T_1: A : = A + 10 ;

T_2: A : = A + A*0.1;

T_3: A : = A -20;

设 A 的初值为 30。

（1） 若这 3 个事务允许并行执行，则有多少可能的正确结果，请一一列举出来。

答：因为串行执行次序有：$T_1T_2T_3$，$T_1T_3T_2$，$T_2T_1T_3$，$T_2T_3T_1$，$T_3T_2T_1$，$T_3T_1T_2$，对应的执行结果 24、22、23、23、21、22，因此 3 个事务并行执行共有四种可能的正确结果，分别是 24、22、23、21。

（2） 请给出一个可串行化的调度，并给出执行结果

答：执行结果为 24

T1	T2	T3
XLOCK(A)		
READ(A)		
A:=A+10	XLOCK(A)	XLOCK(A)
WRITE(A)	…	..
UNLOCK(A)	…	..
	READ(A)	..
	A:=A+A*0.1	..
	WRITE(A)	..
	UNLOCK(A)	..
		..
		READ(A)
		A:=A-20
		WRITE(A)
		UNLOCK(A)

（3） 请给出一个非串行化的调度，并给出执行结果。

答：最后执行结果为 22。

T1	T2	T3
Slock(A)		
Y=A=30		
Unlock(A)		
	Slock(A)	
	Y=A=30	
Xlock(A)		
等待	Unlock(A)	
A=Y+10		
写回 A (=40)		Slock(A)
Unlock(A)		等待
		Y=A=40
		Unlock(A)
	Xlock(A)	Xlock(A)
	等待	
	等待	Y=Y-20
	等待	写回 A (=20)
	等待	Unlock (A)
	A=Y+Y*0.1	
	写回 A (=22)	
	Unlock(A)	

(4) 若这 3 个事务都遵守两段锁协议，请给出一个不产生死锁的可串行化调度。

T1	T2	T3
Slock(A)		
Y=A=30		
Xlock(A)		
A=Y+10	Slock(A)	
写回 A（=40）	等待	
Unlock(A)	等待	
	Y=A=40	
	Xlock(A)	
Unlock(A)	等待	
	A=Y+Y*0.1	Slock(A)
	写回 A（=44）	等待
	Unlock(A)	等待
		等待
		等待
	Unlock(A)	Y=A=44
		Xlock(A)
		Y=Y-20
		写回 A（=24）
		Unlock（A）
		Unlock（A）

(5) 若这 3 个事务都遵守两段锁协议，请给出一个产生死锁的调度。

T1	T2	T3
Slock(A)		
Y=A=30		
	Slock(A)	
	Y=A=30	
Xlock(A)		
等待		
	Xlock(A)	
	等待	
		Slock(A)
		Y=A=30
		Xlock(A)
		等待

9．对以下两个调度：

```
S1=R1(A)R2(A)R3(B)W1(A)R2(C)R2(B)W2(B)W1(C)
S2=W3(A)R1(A)W1(B)R2(B)W2(C)R3(C)
```

回答如下问题：

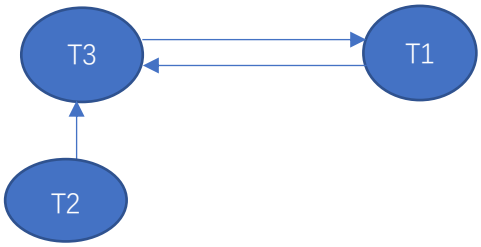
(1) 给出调度的前驱图；

```
S1=R1(A)R2(A)R3(B)W1(A)R2(C)R2(B)W2(B)W1(C)
```

答：


```
S2=W3(A)R1(A)W1(B)R2(B)W2(C)R3(C)
```

答：



(2) 调度是冲突可串性化的吗？如果是，等价的串性调度有哪些？

答：调度 S1 是冲突可串行化的，等价的串行调度有：

```
S1-1= R3(B)R2(A) R2(C)R2(B)W2(B) R1(A)W1(A) W1(C)
```

调度 S2 不是冲突可串行化的。

10．为什么要引进意向锁？简述其含义。

答：DBMS 允许系统同时为不同的事务提供不同的封锁粒度选择，则事务在为某数据对象进行显示封锁的同时，也对其上、下级节点进行了隐式加锁。隐式封锁和显式封锁的效果是一样的，系统检查封锁冲突时两种封锁都要检查。这种搜索效率太低，破坏了当初设立多粒度封锁机制的初衷。为此引入一种新型锁——意向锁。如果对一个节点加意向锁，则意味着要对该节点的所有子孙节点显式加锁；在一个节点显式加锁前，该节点的所有祖先节点都应加上意向锁。因此，事务判定能否给一个节点加锁时不必搜索整棵树。

意向锁的含义是：对任一节点加锁时，必须先对它的上层节点加意向锁。

11．在如下动作序列中加入必要的锁，使事务 T_1 满足：

```
R1(A)R1(B)W1(A)W1(B)
```

（1） 两段锁协议；

XLOCK（A）XLOCK（B）R1(A)R1(B)W1(A) UNLOCK（A）W1(B) UNLOCK（B）

（2） 严格两段锁协议。

XLOCK（A）XLOCK（B）R1(A)R1(B)W1(A) W1(B) UNLOCK（A）UNLOCK（B）

12．假设下面的各个动作序列后面都跟有事务 T_1 的中止动作，说明哪些事务需要回滚。

（1）$S_1=R_1(A)R_2(B)W_1(B)W_2(C)R_3(B)R_3(C)W_3(D)$

（2）$S_2=R_1(A)W_1(B)R_2(B)W_2(C)R_3(C)W_3(D)$

（3）$S_3=R_2(A)R_3(A)R_1(A)W_1(B)R_2(B)R_3(B)W_2(C)R_3(C)$

（4）$S_4=R_2(A)R_3(A)R_1(A)W_1(B)R_3(B)W_2(C)R_3(C)$

答：（1）$S_1=R_1(A)R_2(B)W_1(B)W_2(C)R_3(B)R_3(C)W_3(D)$

没有事务需要回滚

（6） $S_2=R_1(A)W_1(B)R_2(B)W_2(C)R_3(C)W_3(D)$

需要回滚的事务 T_2

（7） $S_3=R_2(A)R_3(A)R_1(A)W_1(B)R_2(B)R_3(B)W_2(C)R_3(C)$

需要回滚的事务有 T_3

（8） $S_4=R_2(A)R_3(A)R_1(A)W_1(B)R_3(B)W_2(C)R_3(C)$

需要回滚的事务有 T3

13．试述常用的意向锁：IS 锁、IX 锁、SIX 锁，给出这些锁的相容矩阵。

答：如果一个节点加 IS 锁，那么将在该节点的子孙节点进行显式封锁，加 S 锁。如果一个节点加 IX 锁，那么表示将在该节点的子孙节点进行显式封锁，可以加排它锁和共享锁。若一个节点加 SIX 锁，那么将对该节点的子节点显式地加共享锁，对更低层的节点显式地加排它锁。

申请锁 \ 已分配的锁 (T2) / (T1)	S	X	IS	IX	SIX
S	Y	N	Y	N	N
X	N	N	N	N	N
IS	Y	N	Y	Y	Y
IX	N	N	Y	Y	N
SIX	N	N	Y	N	N

Y 相容

N 不相容

第 11 章　故 障 恢 复

1．数据库中为什么要有恢复子系统？它的功能是什么？

答：数据库系统不可能不出故障。计算机系统中硬件的故障、软件的错误、操作人员的失误及恶意的破坏都是不可避免的。对付故障除了尽可能提高系统的可靠性外；另一种有效的方式就是在系统发生故障后，利用技术将数据库恢复到故障发生之前的一致状态。因此，恢复子系统必不可少。系统发生故障时，可能会导致数据的丢失，要恢复丢失的数据，就必须事先建立有数据的后备副本。发生故障后，事务回滚，需要利用日志文件中记录的事务的操作。因此，恢复机制需要提供两项功能：一是建立冗余数据；二是利用冗余数据实施数据库恢复。

2．数据库运行中可能产生的故障有哪几类？哪些故障影响事务的正常执行？哪些故障破坏数据库数据？

答：数据库运行中可能产生事务故障、系统故障和介质故障。事务故障和系统故障影响事务的正常执行，介质故障会破坏数据库中的数据。

3．数据库恢复的基本技术有哪些？

答：数据库恢复的基本技术包括：登记日志文件和进行数据备份。

4．数据库转储的意义是什么？试比较各种数据转储方法。

答：数据转储是数据库恢复中采用的基本技术。转储后生成的后备副本保存了数据库在某一时刻的状态，若数据库遭到破坏，就可以装入后备副本将数据库恢复到转储时刻的状态了。当数据库规模很大时，建立备份是一个冗长的过程，通常应尽量避免在每个备份步骤中都要复制整个数据库。因此，存在两个级别的备份。完全转储是备份整个数据库。增量转储只备份上一次备份后发生改变的数据。可以用一个完全转储及其后续的增量转储来恢复数据库。将完全转储复制回数据库，然后以从前向后的顺序，做后续增量转储所记录的改变。由于增量转储一般只包括上次转储以来改变的少量数据，它们占用的空间比完全转储少，做起来也更快。此外，根据转储过程中是否有事务在运行，可以将转储分为静态转储和动态转储。静态转储期间不允许有事务在运行。动态转储期间允许事务对数据库进行更新。大多数数据库不能在做备份时就停止业务的操作，因此，静态转储虽然实现简单，人们还是常用动态转储进行数据库的备份。它不用等待用户执行事务的结束，也不会影响新事务的运行。当动态转储按照某种顺序复制数据库中数据时，有可能正好有事务在对这些数据做更新。因此，复制到备份中的数据有可能不是转储开始的数据。

5．什么是数据库日志文件？为什么要设立日志文件？

答：日志是 DBMS 用来记录事务对数据库的更新操作的文件。数据库系统在运行过程中，除了需要维持业务上的数据一致，还需要在系统崩溃等情况下保证数据的一致性，这就要将事务的状态以及对数据库修改的详细步骤与内存中的数据分开存放，并存储于磁盘等稳定的介质中，在系统故障等的情况下，我们可以通过这些记录来将系统恢复到一致性的状态之下。

6．登记日志文件时为什么必须先写日志文件，后写数据库？

答：把对数据库的修改写到数据库中和把反映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，而日志记录中未反映该修改，则以后无法恢复该修改操作。如果先写日志记录，则无论有没有对数据库进行修改，以后也可以根据日志记录在恢复时多执行一次重做操作或撤销操作，不会影响数据库的正确性。

7．什么是检查点记录？检查点记录包括哪些内容？

答：检查点记录是一类新的日志记录，它的内容包括：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。

8．针对不同的故障，试给出恢复的策略和方法。

答：针对事务故障，恢复策略是利用日志文件撤销此事务对数据库已经做过的修改。若数据后像在事务提交后才写入数据库，则发生故障时数据库中的数据并没有发生变化，所有数据项的修改只是在日志文件中有记录。因此，恢复子系统只需忽略这些未完成的事务就可以了。若事务更新采用的是后像在事务提交前必须写入数据库的方式或在事务提交前后写入数据库的方式，则发生故障时，系统可能已将部分或全部数据项的修改写入磁盘，为保证数据库的一致性，恢复管理器必须使用日志文件撤销此事务对数据库的修改。此时系统恢复的步骤包括：（1）反向扫描日志文件，找出日志文件中该事务的<T,X,V1>记录。（2）执行更新操作的逆操作，将数据库中 X 数据项的值更新为旧值 V_1。（3）继续反向扫描日志文件，找出该事务的其他更新记录，并重复上一步。（4）当扫描到事务的开始标识<T,START>时，故障恢复完成。

针对系统故障，恢复的策略是利用日志文件回滚未完成事务，重做已提交事务。当系统崩溃重新启动时，它构造两个队列：undo-list 存放需要撤销的事务标识符；redo-list 存放需要重做得事务标识符。这两个队列刚开始时都是空的。队列构造步骤为：(1) 系统反向扫描日志，直到发现第一条<CHECKPOINT〉记录。（2）对每一条<T_i,COMMIT>记录，将 T_i加入 redo-list。(3) 对每一条<T_i,START>记录，如果 T_i不属于 redo-list，则将 T_i加入 undo-list。redo-list 和 undo-list 构造完毕，后续的恢复方法又与事务更新执行的方式相关。

（1）后像（AI）在事务提交后才写入数据库。

只要事务在日志中没有<T,COMMIT>记录，我们就知道事务 T 对数据库所做的更新都没有写到磁盘上。因此，恢复时对未完成事务的处理是忽略它们，像它们从未发生过似的。而对提交事务，恢复时将日志记录中的新值再写一次。

系统接下来的恢复步骤如下。

- 对 undo-list 中的事务，在日志中写入一条<T_i,ABORT>记录并刷新日志。
- 对 redo-list 中的事务执行 REDO 操作：从前面发现的<CHECKPOINT>记录开始，正向扫描日志文件，对遇到的每一条<T_i,X,V1>记录，将数据库中的 X 数据项更新为新值 V_1。

（2）后像（AI）在事务提交前必须写入数据库。

如果事务有日志记录<T,COMMIT>，则事务 T 所做的全部改变在此之前已写入磁盘，对这些事务恢复子系统可以忽略。而对未提交事务，恢复子系统根据日志记录做回滚操作。

系统接下来的恢复步骤如下。

- 对 undo-list 中的某一事务，执行 UNDO 操作。
- 再次反向扫描日志文件，对遇到的每一条<T_i,X,V_1>记录，将数据库中的 X 数据项更新为旧值 V_1，扫描到<T_i,START>记录时，扫描停止。
- 在日志中写入一条<T_i,ABORT>记录并刷新日志。
- 重复上两个步骤，直至处理完撤销队列中的每一个事务。

（3）后像（AI）在事务提交前后写入数据库。

在构造完 redo-list 和 undo-list 后，恢复过程继续做如下工作。

- 系统重新反向扫描日志文件，对 undo-list 中的每一事务执行 UNDO 操作，即对遇到的每一条<T_i,X,V_1,V_2>记录，将数据库中的 X 数据项更新为旧值 V_1。
- 在日志中写入一条<T_i,ABORT>记录并刷新日志。当 undo-list 中所有事务 T_i所对应的<T_i, START>记录都找到时，扫描结束。
- 系统找出日志中最后一条<CHECKPOINT>记录。

- 系统由最后一条<CHECKPOINT>记录开始，正向扫描日志文件，对 redo-list 中的事务 T_i的每一条日志记录执行 REDO 操作。即对遇到的每一条<T_i,X,V_1,V_2>记录，将数据库中的 X 数据项更新为新值 V_2。

使用以上步骤的算法，在重做 redo-list 的事务前，撤销 undo-list 中的事务是很重要的。否则，可能会发生问题。

发生介质故障后，磁盘上的物理数据和日志文件数据都有可能被破坏，需要通过日志和最近的备份重建数据库。

（1）装入最近的完全转储后备副本。若数据库副本是动态转储的，还需要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将数据库恢复到某个一致性状态。

（2）如果有后续的增量转储，按照从前往后的顺序，根据增量转储来修改数据库。

（3）装入转储结束后的日志文件副本，重做已完成的事务。即首先反向扫描日志文件，找出故障发生时已经提交的事务，将事务标识符写入 redo-list。然后正向扫描日志文件，对 redo-list 中的所有事务进行 REDO 操作。

这样，就可以将数据库恢复到离故障发生时最近某一时刻的一致性状态了。

9 .设有两个事务 T_1和 T_2的一系列日志记录：<T_1,START>、<T_1,A,22>、<T_2,START>、<T_2,B,28.5>、<T_1,C,61>、<T_2,D,40>、<T_2,COMMIT>、<T_1,F,60>、<T_1,COMMIT>。若发生故障时磁盘上的最后一条日志记录为：

（1）<T_1,C,61>

（2）<T_2,COMMIT>

（3）<T_1,F,60>

（4）<T_1,COMMIT>

试分别说明采用不同的日志文件记录方式，恢复管理器的处理动作。

答：

（1） 最后一条记录为<T_1,C,61>

若日志文件记录方式为数据后像在事务提交后才写入数据库，则恢复管理器忽略两个未完成的事务，什么操作都不做。

若日志文件记录方式采用的是后像在事务提交前必须写入数据库的方式，则恢复管理器：首先反向扫描日志文件，先后找出日志文件中的<T_1,A,22>、<T_2,B,28.5>记录。然后，执行更新操作的逆操作，将数据库中 A 数据项的值更新为旧值 22，B 数据项的值更新为旧值 28.5。当扫描到事务的开始标识< T_1,START>时，故障恢复完成。恢复管理器将<T_1，ABORT>记录写到日志中，并强制刷新日志。

（2） 最后一条记录为<T_2,COMMIT>

若日志文件记录方式为数据后像在事务提交后才写入数据库，则恢复管理器：

① 从后向前扫描日志，将提交 T2 的事务放入队列 redo-list。

② 从日志文件开始处扫描日志。对遇到的每一条<T,X,V1>记录。

- 如果是 T1 事务，则什么也不做。
- 如果是 T2 事务，则将数据项 B、D 写入值 28.5、40。

③在日志中写入一条< T_1,ABORT>记录并刷新日志。

若日志文件记录方式采用的是后像在事务提交前必须写入数据库的方式，则恢复管理器：

若< T_2,COMMIT>已刷新到磁盘上，则认为事务 T2 已提交，恢复管理器对 T2 什么都不做；若记录还在主存中，那么恢复管理器认为事务 T1、T2 未完成。它由后向前扫描日志，将数据库中的数据项 D、C、B、A 依次修改为 40、61、28.5、22。最后，

恢复管理器将< T_1，ABORT>、< T_2，ABORT>记录写到日志中，并强制刷新日志。

(3) 最后一条记录为<T_1,F,60>

若日志文件记录方式为数据后像在事务提交后才写入数据库，则恢复管理器：

① 从后向前扫描日志，将提交 T2 的事务放入队列 redo-list。

② 从日志文件开始处扫描日志。对遇到的每一条<T,X,V1>记录。

- 如果是 T1 事务，则什么也不做。
- 如果是 T2 事务，则将数据项 B、D 写入值 28.5、40。

③在日志中写入一条< T1,ABSORT>记录并刷新日志。

若日志文件记录方式采用的是后像在事务提交前必须写入数据库的方式，则恢复管理器：

对 T2 什么都不做。它由后向前扫描日志，将数据库中的数据项 F、C、A 依次修改为 60、61、22。最后，恢复管理器将<T1，ABORT>记录写到日志中，并强制刷新日志。

(4) 最后一条记录为<T_1,COMMIT>

若日志文件记录方式为数据后像在事务提交后才写入数据库，则恢复管理器：

① 从后向前扫描日志，若<T_1,COMMIT>记录已写入稳定的存储器，则将 T_1、T_2事务都放入队列 redo-list。若<T_1,COMMIT>记录只是写入日志，但还没有写入稳定的存储器，则将 T_2事务都放入队列 redo-list。

② 从日志文件开始处扫描日志。对遇到的每一条<T,X,V1>记录。

- 如果 T_1 为一个未完成的事务，则恢复管理器只需在日志中增加一条<T_1,ABORT>记录。
- 如果 T_1 为一个已完成的事务，则回复管理器为数据项 F、D、C、B 、A 再次写入值 60、40、61、28.5、22。

若日志文件记录方式采用的是后像在事务提交前必须写入数据库的方式，则恢复管理器：

若< T_1,COMMIT>已刷新到磁盘上，则认为事务 T_1已提交，恢复管理器对 T_1 T_2什么都不做；若记录还在主存中，那么恢复管理器认为事务 T1 为未完成事务，T2 为已完成事务。它由后向前扫描日志，将数据库中的数据项 F、C、A 依次修改为 60、61、22。最后，恢复管理器将< T_1，ABORT>记录写到日志中，并强制刷新日志。

第 12 章　数据库安全

1．什么是数据库的安全?
答：数据库的安全是指保护数据库以防止不合法地使用所造成的数据泄漏、更新或破坏。

2．DBMS 提供的安全性控制功能包括哪些内容?
答：DBMS 中提供的数据库安全控制功能包括：
（1）用户标识和鉴别：该方法由系统提供一定的方式让用户标识自己的名字或身份。每次用户要求进入系统时，由系统进行核对，通过鉴定后才提供系统的使用权。
（2）存取控制：通过用户权限定义和合法权限检查确保只有合法权限的用户才能访问数据库，所有未被授权的人员无法存取数据。例如，C2 级中的自主访问控制。
（3）数据分级：为每一数据对象（文件、记录或属性等）赋予一定的保密级，用户也分成类似的级别。系统按规定规则管理用户及用户操作的数据对象。例如，BI 级中的强制访问控制。
（4）视图机制：为不同的用户定义视图，通过视图机制把要保密的数据对无权存取的用户隐藏起来，从而自动地对数据提供一定程度的安全保护。
（5）审计：建立审计日志，把用户对数据库的所有操作自动记录下来放入审计日志中，DBA 可以利用审计跟踪的信息，重现导致数据库现有状况的一系列事件，找出非法存取数据的人、时间和内容等。
（6）数据加密：对存储和传输的数据进行加密处理，从而使得不知道解密算法的人无法获知数据的内容。

3．数据库的安全性和数据库的完整性有什么区别?
答：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，防止错误数据的输入和输出所造成的无效操作和错误结果。
数据的安全性是防止非法用户的非法操作所造成的对数据库的恶意破坏。

4．数据库安全性和计算机系统的安全性有什么关系?
答：安全性问题不是数据库系统所独有的，所有的计算机系统都有这个问题。只是在数据库系统中大量数据集中存放，且为多用户直接共享，因此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指标之一。
数据库安全性和计算机系统的安全性，包括操作系统、网络系统的安全性是紧密联系、相互支持的。

5．简述数据库三种访问控制方法。
答：用户标识和鉴别是数据库系统提供的最外层安全保护措施。其方法是由系统提供一定的方式让用户标识自己的身份或名字。每次用户要求进入系统时，由系统进行核对，通过鉴定后才能提供机器使用权。
自主访问控制是对用户访问数据库中各种资源的权利的控制。数据库用户按其访问权限的大小一般分为具有 CONNECT 特权的用户、具有 RESOURCE 特权的用户及具有 DBA 特权的用户。
强制访问控制为每一个数据对象标以一定的密级，而每个用户也被授予某一个级别的许可证。系统规定只有具有某一许可证级别的用户才能存取某一密级的数据对象。

6．简要阐述影响数据加密的关键因素有哪些?
答：影响数据加密的关键因素包括：加密粒度、加密算法以及密钥管理机制。

7．为什么强制访问控制提供了更高级别的数据库安全性?
答：强制访问控制一般与自主访问控制结合使用，实施一些附加的、更强的访问限制。一个主体只有通过了自主与强制性访问限制检查后，才能访问某个客体。用户可以利用自主访问控制来防范其他用户对自己客体的攻击，由于用户不能直接改变强制访问控制属性，所以强制访问控制提供了一个不可逾越的、更强的安全保护层以防止其他用户偶然或故意地滥用自主访问控制。

8．理解并解释 MAC 机制中主体、客体、敏感度标记的含义。
答：主体是系统中的活动实体，既包括 DBMS 所管理的实际用户，也包括代表用户的各进程。
客体是系统中的被动实体，是受主体操纵的，包括文件、基表、索引、视图等。
对于主体和客体，DBMS 为它们每个实例（值）指派一个敏感度标记。敏感度标记分成若干级别。例如绝密、机密、可信、公开等。主体的秘感度标记称为许可证级别，客体的敏感度级别称为密级。

9．举例说明 MAC 机制如何确定主体能否存取客体?
答：假定系统设置有 4 个等级：Top Secret（TS）、Secret（S）、Confidential（C）、Unclassified（U）。关系表 RecipeMaster 中每一行都有自己的安全等级，见下表。
若用户 LiXia 和 WangHao 的许可证级别分别是 C 和 S，则遵循保密性规则用户 LiXia 看不到表中的任何记录，而 WangHao 可以看到其中的一条记录。

表

Rno	Pno	Dno	PGno	Rdatetime	Security class
1282317	481	140	1645	2007-07-21 13:12:01	S
1282872	201	21	2170	2007-07-22 10:10:03	TS

10．　什么是数据库的审计功能？为什么要提供审计功能?
答：审计功能是指 DBMS 的审计模块在用户对数据库执行操作的同时把所有操作自动记录到系统的审计日志中。
由于所有的保密措施都不是绝对可靠的，攻击者总有办法突破这些控制，只是付出的代价大小而已。利用数据库的跟踪审计功能系统可自动报警，DBA 也可以根据审计跟踪的信息，重现导致数据库现有状态的一系列事件，找出非法存取数据的人、时间和内容等，根据数据进行事后的分析和调查。