第八章　并 发 控 制

事务处理技术主要包括数据库恢复技术和并发控制技术。本章讨论数据库并发控制的基本概念和实现技术。本章内容有一定的深度和难度。读者学习本章一定要做到概念清楚。

为此要认真读书,特别是读书上的例题。读书时要自己动手先做一做例题,体会一下是否掌握了有关概念。

一、基本知识点

数据库是一个共享资源,当多个用户并发存取数据库时就会产生多个事务同时存取同一个数据的情况。若对并发操作不加控制就可能会存取和存储不正确的数据,破坏数据库的一致性。所以 DBMS 必须提供并发控制机制。

并发控制机制的正确性和高效性是衡量一个 DBMS 性能的重要标志之一。

① 需要了解的:数据库并发控制技术的必要性,活锁死锁的概念。

② 需要牢固掌握的:并发操作可能产生数据不一致性的情况(丢失修改、不可重复读、读“脏数据”)及其确切含义;封锁的类型;不同封锁类型的(例如 X 锁,S 锁)的性质和定义,相关的相容控制矩阵;封锁协议的概念;封锁粒度的概念;多粒度封锁方法;多粒度封锁协议的相容控制矩阵。

③ 需要举一反三的:封锁协议与数据一致性的关系;并发调度的可串行性概念;两段锁协议与可串行性的关系;两段锁协议与死锁的关系。

④ 难点:两段锁协议与串行性的关系;与死锁的关系;具有意向锁的多粒度封锁方法的封锁过程。

二、习题解答和解析

1. 在数据库中为什么要并发控制?

答

数据库是共享资源,通常有许多个事务同时在运行。

当多个事务并发地存取数据库时就会产生同时读取和/或修改同一数据的

情况。若对并发操作不加控制就可能会存取和存储不正确的数据,破坏数据库的一致性。所以数据库管理系统必须提供并发控制机制。

2. 并发操作可能会产生哪几类数据不一致?用什么方法能避免各种不一致的情况?

答

并发操作带来的数据不一致性包括三类:丢失修改、不可重复读和读"脏"数据。

(1) 丢失修改(Lost Update)

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(覆盖了)T_1 提交的结果,导致 T_1 的修改被丢失。

(2) 不可重复读(Non - Repeatable Read)

不可重复读是指事务 T_1 读取数据后,事务 T_2 执行更新操作,使 T_1 无法再现前一次读取结果。不可重复读包括三种情况:详见《概论》8.1(P266)。

(3) 读"脏"数据(Dirty Read)

读"脏"数据是指事务 T_1 修改某一数据,并将其写回磁盘,事务 T_2 读取同一数据后,T_1 由于某种原因被撤销,这时 T_1 已修改过的数据恢复原值,T_2 读到的数据就与数据库中的数据不一致,则 T_2 读到的数据就为"脏"数据,即不正确的数据。

避免不一致性的方法和技术就是并发控制。最常用的技术是封锁技术。也可以用其他技术,例如在分布式数据库系统中可以采用时间戳方法来进行并发控制。

3. 什么是封锁?

答

封锁就是事务 T 在对某个数据对象例如表、记录等操作之前,先向系统发出请求,对其加锁。加锁后事务 T 就对该数据对象有了一定的控制,在事务 T 释放它的锁之前,其他的事务不能更新此数据对象。

封锁是实现并发控制的一个非常重要的技术。

4. 基本的封锁类型有几种?试述它们的含义。

答

基本的封锁类型有两种:排它锁(Exclusive Locks,简称 X 锁)和共享锁(Share Locks,简称 S 锁)。

排它锁又称为写锁。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。

共享锁又称为读锁。若事务 T 对数据对象 A 加上 S 锁,则事务 T 可以读 A

但不能修改 A,其他事务只能再对 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 S 锁之前不能对 A 做任何修改。

5. 如何用封锁机制保证数据的一致性?

答

DBMS 在对数据进行读、写操作之前首先对该数据执行封锁操作,例如下图中事务 T_1 在对 A 进行修改之前先对 A 执行 Xock(A),即对 A 加 X 锁。这样,当 T_2 请求对 A 加 X 锁时就被拒绝,T_2 只能等待 T_1 释放 A 上的锁后才能获得对 A 的 X 锁,这时它读到的 A 是 T_1 更新后的值,再按此新的 A 值进行运算。这样就不会丢失 T_1 的更新。

T_1	T_2
① Xlock A	
获得	
② 读 A = 16	
	Xlock A
③ A←A - 1	等待
写回 A = 15	等待
Commit	等待
Unlock A	等待
④	获得 Xlock A
	读 A = 15
	A←A - 1
	写回 A = 14
⑤	Commit
	Unlock A

DBMS 按照一定的封锁协议,对并发操作进行控制,使得多个并发操作有序地执行,就可以避免丢失修改、不可重复读和读"脏"数据等数据不一致性。

6. 什么是封锁协议?不同级别的封锁协议的主要区别是什么?

答

在运用封锁技术对数据加锁时,要约定一些规则。例如,在运用 X 锁和 S 锁对数据对象加锁时,要约定何时申请 X 锁或 S 锁、何时释放封锁等。这些约定或者规则称为封锁协议(Locking Protocol)。对封锁方式约定不同的规则,就形成了各种不同的封锁协议、不同级别的封锁协议,例如《概论》8.3 中介绍的三级封锁协议,三级协议的主要区别在于什么操作需要申请封锁,何时申请封锁以及何时释放锁(即持锁时间的长短)。

一级封锁协议:事务 T 在修改数据 R 之前必须先对其加 X 锁,直到事务结束才释放。

二级封锁协议:一级封锁协议加上事务 T 在读取数据 R 之前必须先对其加

S 锁,读完后即可释放 S 锁。

三级封锁协议:一级封锁协议加上事务 T 在读取数据 R 之前必须先对其加 S 锁,直到事务结束才释放。

7. 不同封锁协议与系统一致性级别的关系是什么?

答

不同的封锁协议对应不同的一致性级别。

一级封锁协议可防止丢失修改,并保证事务 T 是可恢复的。在一级封锁协议中,对读数据是不加 S 锁的,所以它不能保证可重复读和不读"脏"数据。

二级封锁协议除防止了丢失修改,还可进一步防止读"脏"数据。在二级封锁协议中,由于读完数据后立即释放 S 锁,所以它不能保证可重复读。

在三级封锁协议中,无论是读数据还是写数据都加长锁,即都要到事务结束时才释放封锁。所以三级封锁协议除防止了丢失修改和不读"脏"数据外,还进一步防止了不可重复读。

下面的表格清楚地说明了封锁协议与系统一致性的关系。

	X 锁		S 锁		一致性保证		
	操作结束释放	事务结束释放	操作结束释放	事务结束释放	不丢失修改	不读"脏"数据	可重复读
一级封锁协议		√			√		
二级封锁协议		√	√		√	√	
三级封锁协议		√		√	√	√	√

8. 什么是活锁?什么是死锁?

答

T_1	T_2	T_3	T_4
lock R	·	·	·
·	lock R	·	·
·	等待	Lock R	·
Unlock	等待	·	Lock R
·	等待	Lock R	等待
·	等待	·	等待
·	等待	Unlock	等待
·	等待	·	Lock R
·	等待	·	·

如果事务 T_1 封锁了数据 R,事务 T_2 又请求封锁 R,于是 T_2 等待。T_3 也请求封锁 R,当 T_1 释放了 R 上的封锁之后系统首先批准了 T_3 的请求,T_2 仍然等待。然后 T_4 又请求封锁 R,当 T_3 释放了 R 上的封锁之后系统又批准了 T_4 的请求……T_2 有可能永远等待,这就是活锁的情形。活锁的含义是该等待事务等待时

间太长,似乎被锁住了,实际上可能被激活。

如果事务 T_1 封锁了数据 R_1,T_2 封锁了数据 R_2,然后 T_1 又请求封锁 R_2,因 T_2 已封锁了 R_2,于是 T_1 等待 T_2 释放 R_2 上的锁。接着 T_2 又申请封锁 R_1,因 T_1 已封锁了 R_1,T_2 也只能等待 T_1 释放 R_1 上的锁。这样就出现了 T_1 在等待 T_2,而 T_2 又在等待 T_1 的局面,T_1 和 T_2 两个事务永远不能结束,形成死锁。

T_1	T_2
lock R_1	·
·	Lock R_2
·	·
Lock R_2	·
等待	·
等待	Lock R_1
等待	等待

9. 试述活锁的产生原因和解决方法。

答

活锁产生的原因:当一系列封锁不能按照其先后顺序执行时,就可能导致一些事务无限期等待某个封锁,从而导致活锁。

避免活锁的简单方法是采用先来先服务的策略。当多个事务请求封锁同一数据对象时,封锁子系统按请求封锁的先后次序对事务排队,数据对象上的锁一旦释放就批准申请队列中第一个事务获得锁。

10. 请给出预防死锁的若干方法。

答

在数据库中,产生死锁的原因是两个或多个事务都已封锁了一些数据对象,然后又都请求已被其他事务封锁的数据加锁,从而出现死等待。

防止死锁的发生其实就是要破坏产生死锁的条件。预防死锁通常有两种方法:

(1) 一次封锁法,要求每个事务必须一次将所有要使用的数据全部加锁,否则就不能继续执行;

(2) 顺序封锁法,预先对数据对象规定一个封锁顺序,所有事务都按这个顺序实行封锁。

不过,预防死锁的策略不大适合数据库系统的特点,具体原因可参见《概论》8.4。

11. 请给出检测死锁发生的一种方法,当发生死锁后如何解除死锁?

答

数据库系统一般采用允许死锁发生,DBMS 检测到死锁后加以解除的方法。

DBMS 中诊断死锁的方法与操作系统类似,一般使用超时法或事务等待图法。

超时法是:如果一个事务的等待时间超过了规定的时限,就认为发生了死锁。超时法实现简单,但有可能误判死锁,事务因其他原因长时间等待超过时限时,系统会误认为发生了死锁。若时限设置得太长,又不能及时发现死锁发生。

DBMS 并发控制子系统检测到死锁后,就要设法解除。通常采用的方法是选择一个处理死锁代价最小的事务,将其撤消,释放此事务持有的所有锁,使其他事务得以继续运行下去。当然,对撤销的事务所执行的数据修改操作必须加以恢复。

12. 什么样的并发调度是正确的调度?

答

可串行化(Serializable)的调度是正确的调度。

可串行化的调度的定义:多个事务的并发执行是正确的,当且仅当其结果与按某一次序串行执行它们时的结果相同,称这种调度策略为可串行化的调度。

13. 设 T_1,T_2,T_3 是如下的 3 个事务:

T_1:A:= A + 2;

T_2:A:= A * 2;

T_3:A:= A * * 2; (A A^2)

设 A 的初值为 0。

(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

A 的最终结果可能有 2、4、8、16。

因为串行执行次序有 T_1 T_2 T_3、T_1 T_3 T_2、T_2 T_1 T_3、T_2 T_3 T_1、T_3 T_1 T_2、T_3 T_2 T_1。

对应的执行结果是 16、8、4、2、4、2。

(2) 请给出一个可串行化的调度,并给出执行结果

答

T_1	T_2	T_3
Slock A		
Y = A = 0		
Unlock A		
Xlock A		
	Slock A	
A = Y + 2	等待	
写回 A(= 2)	等待	
Unlock A	等待	

Y = A = 2	
Unlock A	
Xlock A	
	Slock A
A = Y * 2	等待
写回 A(= 4)	等待
Unlock A	等待
	Y = A = 4
	Unlock A
	Xlock A
	A = Y * *2
	写回 A(= 16)
	Unlock A

最后结果 A 为 16,是可串行化的调度。

(3) 请给出一个非串行化的调度,并给出执行结果。

答

T_1	T_2	T_3
Slock A		
Y = A = 0		
Unlock A		
	Slock A	
	Y = A = 0	
Xlock A		
等待	Unlock A	
A = Y + 2		
写回 A(= 2)		Slock A
Unlock A		等待
		Y = A = 2
		Unlock A
		Xlock A
	Xlock A	
	等待	Y = Y * *2
	等待	写回 A(= 4)
	等待	Unlock A
	A = Y * 2	
	写回 A(= 0)	
	Unlock A	

最后结果 A 为 0,为非串行化的调度。

(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。

答

T_1	T_2	T_3
Slock A		
Y = A = 0		
Xlock A		
A = Y + 2	Slock A	
写回 A(= 2)	等待	
Unlock A	等待	
	Y = A = 2	
	Xlock A	
Unlock A	等待	Slock A
	A = Y * 2	等待
	写回 A(= 4)	等待
	Unlock A	等待
		Y = A = 4
	Unlock A	
		Xlock A
		A = Y * *2
		写回 A(= 16)
		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。

答

T_1	T_2	T_3
Slock A		
Y = A = 0		
	Slock A	
	Y = A = 0	
Xlock A		
等待		
	Xlock A	
	等待	
		Slock A
		Y = A = 0
		Xlock A
		等待

14. 试述两段锁协议的概念。

答

两段锁协议是指所有事务必须分两个阶段对数据项加锁和解锁。

在对任何数据进行读、写操作之前,首先要申请并获得对该数据的封锁;

在释放一个封锁之后,事务不再申请和获得任何其他封锁。

“两段”的含义是,事务分为两个阶段:

第一阶段是获得封锁,也称为扩展阶段,在这阶段,事务可以申请获得任何数据项上的任何类型的锁,但是不能释放任何锁;

第二阶段是释放封锁,也称为收缩阶段,在这阶段,事务释放已经获得的锁,但是不能再申请任何锁。

15. 试证明,若并发事务遵守两段锁协议,则对这些事务的并发调度是可串行化的。

证明

首先以两个并发事务 T_1 和 T_2 为例,存在多个并发事务的情形可以类推。

根据可串行化定义可知,事务不可串行化只可能发生在下列两种情况:

(1) 事务 T_1 写某个数据对象 A,T_2 读或写 A;

(2) 事务 T_1 读或写某个数据对象 A,T_2 写 A。

下面称 A 为潜在冲突对象。

设 T_1 和 T_2 访问的潜在冲突的公共对象为$\{A_1, A_2, \cdots, A_n\}$。

不失一般性,假设这组潜在冲突对象中 $X = \{A_1, A_2, \cdots, A_i\}$均符合情况 1。$Y = \{A_{i+1}, \cdots, A_n\}$符合所情况(2)。

"x∈X,T_1 需要 Xlock x ①

T_2 需要 Slock x 或 Xlock x ②

1) 如果操作①先执行,则 T_1 获得锁,T_2 等待

由于遵守两段锁协议,T_1 在成功获得 X 和 Y 中全部对象及非潜在冲突对象的锁后,才会释放锁。

这时如果v w∈X 或 Y,T_2 已获得 w 的锁,则出现死锁;

否则,T_1 在对 X、Y 中对象全部处理完毕后,T_2 才能执行。

这相当于按 T_1、T_2 的顺序串行执行,根据可串行化定义,T_1 和 T_2 的调度是可串行化的。

2) 操作②先执行的情况与(1)对称

因此,若并发事务遵守两段锁协议,在不发生死锁的情况下,对这些事务的并发调度一定是可串行化的。

证毕。

16. 举例说明,对并发事务的一个调度是可串行化的,而这些并发事务不一定遵守两段锁协议。

T_1	T_2
Slock B	
读 B = 2	
Y = B	
Unlock B	
Xlock A	
	Slock A
A = Y + 1	等待
写回 A = 3	等待

Unlock A	等待
	等待
	Slock A
	读 A = 3
	X = A
	Unlock A
	Xlock B
	B = X + 1
	写回 B = 4
	Unlock B

17. 为什么要引进意向锁？意向锁的含义是什么？

答

引进意向锁是为了提高封锁子系统的效率。该封锁子系统支持多种封锁粒度。

原因是：在多粒度封锁方法中一个数据对象可能以两种方式加锁——显式封锁和隐式封锁(有关概念参见《概论》8 7.1)。因此系统在对某一数据对象加锁时不仅要检查该数据对象上有无(显式和隐式)封锁与之冲突，还要检查其所有上级结点和所有下级结点，看申请的封锁是否与这些结点上的(显式和隐式)封锁冲突，显然，这样的检查方法效率很低。为此引进了意向锁。

意向锁的含义是：对任一结点加锁时，必须先对它的上层结点加意向锁。

例如事务 T 要对某个元组加 X 锁，则首先要对关系和数据库加 IX 锁。换言之，对关系和数据库加 IX 锁，表示它的后裔结点——某个元组拟(意向)加 X 锁。

引进意向锁后，系统对某一数据对象加锁时不必逐个检查与下一级结点的封锁冲突了。例如，事务 T 要对关系 R 加 X 锁时，系统只要检查根结点数据库和 R 本身是否已加了不相容的锁(如发现已经加了 IX，则与 X 冲突)，而不再需要搜索和检查 R 中的每一个元组是否加了 X 锁或 S 锁。

18. 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

IS 锁

如果对一个数据对象加 IS 锁，表示它的后裔结点拟(意向)加 S 锁。例如，要对某个元组加 S 锁，则要首先对关系和数据库加 IS 锁

IX 锁

如果对一个数据对象加 IX 锁，表示它的后裔结点拟(意向)加 X 锁。例如，要对某个元组加 X 锁，则要首先对关系和数据库加 IX 锁。

SIX 锁

如果对一个数据对象加 SIX 锁，表示对它加 S 锁，再加 IX 锁，即 SIX = S + IX。

相容矩阵

T_1 \ T_2	S	X	IS	IX	SIX	-
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
-	Y	Y	Y	Y	Y	Y

19. 理解并解释下列术语的含义:封锁、活锁、死锁、排它锁、共享锁、并发事务的调度、可串行化的调度、两段锁协议。

答

(略,已经在上面有关习题中解答)

*20. 试述你了解的某一个实际的 DBMS 产品的并发控制机制。

答

(略,参见《概论》8.8 节,简单介绍了有关 Oracle 的并发控制机制。)