

数据库管理2

并发控制

授课教师：秦建斌

邮箱：qinjianbin@szu.edu.cn

深圳大学 计算机与软件学院

目录

- 事务
- 并发控制
- 封锁
- 并发调度的可串行性
- 两段锁协议
- 封锁粒度

15分

1.事务

□事务(Transaction)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。

□事务是数据库的逻辑工作单位

□事务和程序是两个概念

■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

■一个程序通常包含多个事务

□事务是恢复和并发控制的基本单位

部分做会丢失数据

定义事务

显式定义方式

BEGIN TRANSACTION

SQL 语句1

SQL 语句2

○ ○ ○ ○ ○

COMMIT

BEGIN TRANSACTION

SQL 语句1

SQL 语句2

○ ○ ○ ○ ○

ROLLBACK

- 事务异常终止
- 事务运行的过程
- 系统将事务中对数据库的所有更新撤消
- 事务滚回到开始时的状态

- 事务正常结束
- 提交事务的所有操作（读+更新）
- 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

121 系统

事务的特性（ACID特性）

事务的ACID特性：

□ 原子性（Atomicity）

- 事务中包括的诸操作要么都做，要么都不做

□ 一致性（Consistency）

- 数据库中只包含成功事务提交的结果

□ 隔离性（Isolation）

- 一个事务的执行不能被其他事务干扰

□ 持续性（Durability）

- 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。

2.并发控制（对应第11章）

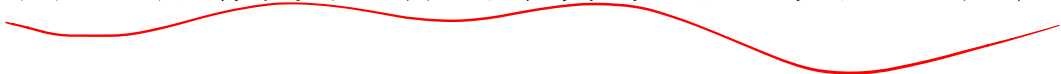
□多用户数据库系统

允许多个用户同时使用的数据库系统

■飞机定票数据库系统

■银行数据库系统

■特点：在同一时刻并发运行的事务数可达数百上千个

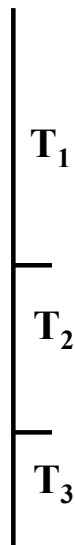


并发控制（续）

□多事务执行方式

（1）事务串行执行

- 每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行
- 不能充分利用系统资源，发挥数据库共享资源的特点

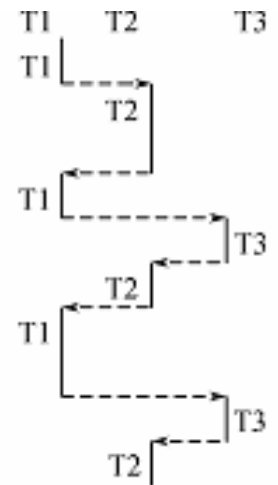


事务的串行执行方式

并发控制（续）

（2）交叉并发方式（Interleaved Concurrency）

- 在单处理机系统中，事务的并行执行是这些并行事务的并行操作轮流交叉运行
- 单处理机系统中的并行事务并没有真正地并行运行，但能够减少处理机的空闲时间，提高系统的效率



(b) 事务的交叉并发执行方式

并发控制（续）

（3）同时并发方式（simultaneous concurrency）

- 多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务真正的并行运行

- 最理想的并发方式，但受制于硬件环境

- 更复杂的并发方式机制

- 本章讨论的数据库系统并发控制技术是以单处理机系统为基础的

并发控制（续）

□事务并发执行带来的问题

- 会产生多个事务同时存取同一数据的情况
- 可能会存取和存储不正确的数据，破坏事务隔离性和数据库的一致性

□数据库管理系统必须提供并发控制机制

□并发控制机制是衡量一个数据库管理系统性能的重要标志之一

并发控制概述（续）

□ 并发操作带来的数据不一致性

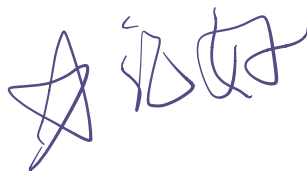
- (1) 丢失修改 (Lost Update)
- (2) 不可重复读 (Non-repeatable Read)
- (3) 读“脏”数据 (Dirty Read)



□ 记号

■ $R(x)$: 读数据 x

■ $W(x)$: 写数据 x



(1) 丢失修改

❑ 两个事务 T_1 和 T_2 读入同一数据并修改， T_2 的提交结果破坏了 T_1 提交的结果，导致 T_1 的修改被丢失。

❑ 上面飞机订票例子就属

T_1	T_2
① $R(A)=16$	
②	$R(A)=16$
③ $A \leftarrow A-1$ $W(A)=15$	
④	$A \leftarrow A-1$ $W(A)=15$

(2) 不可重复读

□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：数据相关。

(1) 事务 T_1 读取某一数据后，事务 T_2 对其做了修改，或事务 T_2 删除了其中部分记录，或事务 T_2 插入了一些记录，当事务 T_1 再次读该数据时，得到与前一次不同的值

不可重复读（续）

例如：

T_1	T_2
① $R(A)=50$	
$R(B)=100$	
求和=150	
②	$R(B)=100$
	$B \leftarrow B * 2$
	$W(B)=200$
③ $R(A)=50$	
$R(B)=200$	
求和=250	
(验算不对)	

不可重复读

■ T_1 读取 $B=100$ 进行运算

■ T_2 读取同一数据 B ，对其进行修改后将 $B=200$ 写回数据库。

■ T_1 为了对读取值校对重读 B ， B 已为 200，与第一次读取值不一致

(3) 读“脏”数据

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
- T_2 读到的数据就为“脏”数据，即不正确的数据

读“脏”数据（续）

例如

T_1	T_2
① $R(C)=100$	
$C \leftarrow C * 2$	
$W(C)=200$	
②	$R(C)=200$
③ ROLLBACK	回滚
C 恢复为100	

读“脏”数据

- T_1 将C值修改为200， T_2 读到C为200
- T_1 由于某种原因撤销，其修改作废，C恢复原值100
- 这时 T_2 读到的C为200，与数据库内容不一致，就是“脏”数据

并发控制概述（续）

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

□ 并发控制的主要技术

■ 封锁(Locking)

3.什么是封锁

□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，其它的事务不能更新此数据对象。

□基本封锁类型

■排它锁（Exclusive Locks，简记为X锁）不能互斥

■共享锁（Share Locks，简记为S锁）可看可用不可改

排它锁

- 排它锁又称为写锁 (Write Locks) X 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，其它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
- 保证其他事务在T释放A上的锁之前不能再读取和修改A

共享锁

□ 共享锁又称为读锁 (Read Locks) S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，其它事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁

□ 保证其他事务可以读A，但在T释放A上的S锁之前不能对A做任何修改

使用封锁机制解决丢失修改问题

例：

T_1	T_2
① Xlock A	
② R(A)=16	
	Xlock A
③ $A \leftarrow A-1$	等待
W(A)=15	等待
Commit	等待
Unlock A	等待
④	获得Xlock A
	R(A)=15
	$A \leftarrow A-1$
⑤	W(A)=14
	Commit
	Unlock A

没有丢失修改

- 事务 T_1 在读A进行修改之前先对A加X锁
- 当 T_2 再请求对A加X锁时被拒绝
- T_2 只能等待 T_1 释放A上的锁后获得对A的X锁
- 这时 T_2 读到的A已经是 T_1 更新过的值15
- T_2 按此新的A值进行运算，并将结果值A=14写回到磁盘。避免了丢失 T_1 的更新。

使用封锁机制解决不可重复读问题

T_1	T_2
① Slock A	
Slock B	
$R(A)=50$	
$R(B)=100$	
求和=150	
②	Xlock B
	等待
③ $R(A)=50$	等待
$R(B)=100$	等待
求和=150	等待
Commit	等待
Unlock A	等待
Unlock B	等待
④	获得XlockB
	$R(B)=100$
	$B \leftarrow B * 2$
⑤	$W(B)=200$
	Commit
	Unlock B

可重复读

- 事务 T_1 在读A, B之前, 先对A, B加S锁
- 其他事务只能再对A, B加S锁, 而不能加X锁, 即其他事务只能读A, B, 而不能修改
- 当 T_2 为修改B而申请对B的X锁时被拒绝只能等待 T_1 释放B上的锁
- T_1 为验算再读A, B, 这时读出的B仍是100, 求和结果仍为150, 即可重复读
- T_1 结束才释放A, B上的S锁。 T_2 才获得对B的X锁

使用封锁机制解决读“脏”数据问题

例

T_1	T_2
① Xlock C	
R(C)=100	
$C \leftarrow C * 2$	
W(C)=200	
②	Slock C
	等待
③ROLLBACK	等待
(C恢复为100)	等待
Unlock C	等待
④	获得Slock C
	R(C)=100
⑤	Commit C
	Unlock C

不读“脏”数据

- 事务 T_1 在对C进行修改之前，先对C加X锁，修改其值后写回磁盘
- T_2 请求在C上加S锁，因 T_1 已在C上加了X锁， T_2 只能等待
- T_1 因某种原因被撤销，C恢复为原值100
- T_1 释放C上的X锁后 T_2 获得C上的S锁，读C=100。避免了 T_2 读“脏”数据

活锁和死锁

- 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。
- 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（续）

T ₁	T ₂	T ₃	T ₄
Lock R	•	•	•
	•	•	•
	•	•	•
•	Lock R		
•	等待	Lock R	
•	等待	•	Lock R
Unlock R	等待	•	等待
	等待	Lock R	等待
•	等待	•	等待
•	等待	Unlock	等待
•	等待	•	Lock R
	等待	•	•
			•

活锁（续）

□避免活锁：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

■按请求封锁的先后次序对这些事务排队

■该数据对象上的锁一旦释放，首先批准申请队列中第一个事务获得锁

死锁

- 事务 T_1 封锁了数据 R_1
- T_2 封锁了数据 R_2
- 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 T_1 在等待 T_2 ，而 T_2 又在等待 T_1 ， T_1 和 T_2 两个事务永远不能结束，形成死锁

死锁（续）

T_1	T_2
Lock R_1	•
	•
	•
•	Lock R_2
•	•
•	•
Lock R_2	•
等待	
等待	
等待	Lock R_1
等待	等待
等待	等待
	•
	•
	•

(b) 死锁

解决死锁的方法

两类方法

1. 死锁的预防

■ 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

■ 顺序封锁法是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。

2. 死锁的诊断与解除

(1) 超时法

(2) 等待图法

(1) 超时法

- 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
- 优点：实现简单
- 缺点
 - 有可能误判死锁
 - 时限若设置得太长，死锁发生后不能及时发现

(2) 等待图法

□用事务等待图动态反映所有事务的等待情况

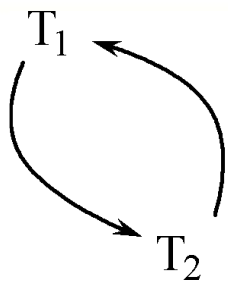
■事务等待图是一个有向图 $G=(T, U)$

■ T 为结点的集合，每个结点表示正运行的事务

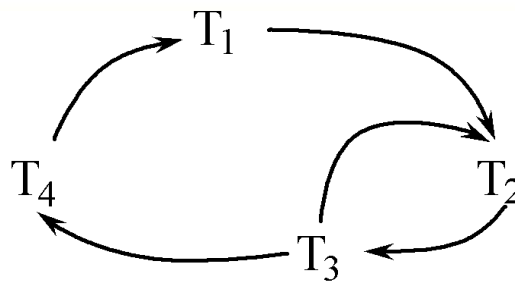
■ U 为边的集合，每条边表示事务等待的情况

■若 T_1 等待 T_2 ，则 T_1, T_2 之间划一条有向边，从 T_1 指向 T_2

等待图法（续）



(a)

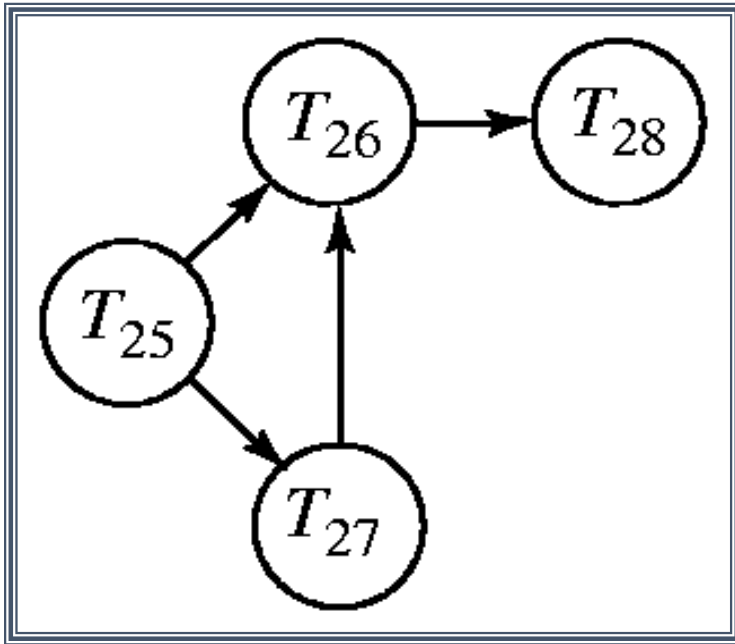


(b)

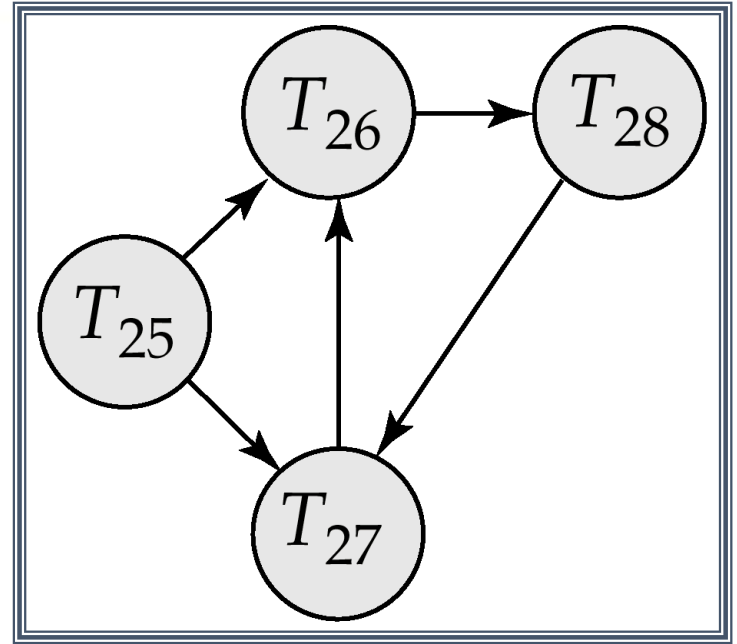
事务等待图

- 图(a)中，事务 T_1 等待 T_2 ， T_2 等待 T_1 ，产生了死锁
 - 图(b)中，事务 T_1 等待 T_2 ， T_2 等待 T_3 ， T_3 等待 T_4 ， T_4 又等待 T_1 ，产生了死锁
 - 图(b)中，事务 T_3 可能还等待 T_2 ，在大回路中又有小的回路
- 如果发现图中存在回路，则表示系统中出现了死锁。

那个发生了死锁??



Wait-for graph without a cycle



Wait-for graph with a cycle

死锁的诊断与解除（续）

□解除死锁

- 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的锁，使其它事务能继续运行下去

4.并发调度的可串行性

- ❑数据库管理系统对并发事务不同的调度可能会产生不同的结果
- ❑串行调度是正确的
- ❑执行结果等价于串行调度的调度也是正确的，称为可串行化调度

可串行化调度

□可串行化(Serializable)调度

- 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地执行这些事务时的结果相同

□可串行性(Serializability)

- 是并发事务正确调度的准则
- 一个给定的并发调度，当且仅当它是可串行化的，才认为是正确调度

可串行化调度（续）

[例11.2]现在有两个事务，分别包含下列操作：

■事务T1：读B； $A=B+1$ ；写回A

■事务T2：读A； $B=A+1$ ；写回B

现给出对这两个事务不同的调度策略

串行调度,正确的调度

T_1	T_2
Slock B	
Y=R(B)=2	
Unlock B	
Xlock A	
A=Y+1=3	
W(A)	
Unlock A	
	Slock A
	X=R(A)=3
	Unlock A
	Xlock B
	B=X+1=4
	W(B)
	Unlock B

串行调度(a)

- 假设A、B的初值均为2。
- 按 $T_1 \rightarrow T_2$ 次序执行结果为A=3, B=4
- 串行调度策略,正确的调度

串行调度,正确的调度

T_1	T_2
	Slock A
	X=R(A)=2
	Unlock A
	Xlock B
	B=X+1=3
	W(B)
	Unlock B
Slock B	
Y=R(B)=3	
Unlock B	
Xlock A	
A=Y+1=4	
W(A)	
Unlock A	

串行调度(b)

- 假设A、B的初值均为2。
- $T_2 \rightarrow T_1$ 次序执行结果为
B=3, A=4
- 串行调度策略,正确的调度

不可串行化调度，错误的调度

T_1	T_2
Slock B	
$Y=R(B)=2$	
	Slock A
	$X=R(A)=2$
Unlock B	
	Unlock A
Xlock A	
$A=Y+1=3$	
$W(A)$	
	Xlock B
	$B=X+1=3$
	$W(B)$
Unlock A	
	Unlock B

- 执行结果与(a)、(b)的结果都不同
- 是错误的调度

不可串行化的调度

可串行化调度， 正确的调度

T_1	T_2
Slock B	
$Y=R(B)=2$	
Unlock B	
Xlock A	
	Slock A
$A=Y+1=3$	等待
$W(A)$	等待
Unlock A	等待
	$X=R(A)=3$
	Unlock A
	Xlock B
	$B=X+1=4$
	$W(B)$
	Unlock B

可串行化的调度

- 执行结果与串行调度 (a) 的执行结果相同
- 是正确的调度

冲突可串行化调度

□如何判断调度是可串行化的调度？？

□冲突可串行化

- 一个比可串行化更严格的条件

- 商用系统中的调度器采用

□冲突操作：是指不同的事务对同一数据的读写操作和写写操作：

$R_i(x)$ 与 $W_j(x)$ /*事务 T_i 读 x , T_j 写 x , 其中 $i \neq j$ */

$W_i(x)$ 与 $W_j(x)$ /*事务 T_i 写 x , T_j 写 x , 其中 $i \neq j$ */

其他操作是不冲突操作

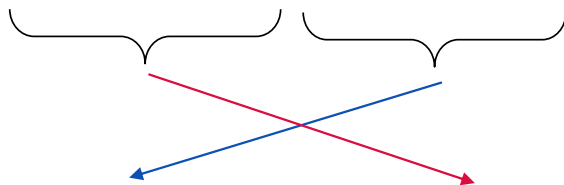
冲突可串行化（续）

- 一个调度 S_c 在保证冲突操作的次序不变的情况下，通过交换两个事务不冲突操作的次序得到另一个调度 S_c' ，如果 S_c' 是串行的，称调度 S_c 是冲突可串行化的调度
- 若一个调度是冲突可串行化，则一定是可串行化的调度
- 可用这种方法判断一个调度是否是冲突可串行化的
- 不能交换（Swap）的动作：
 - 同一事务的两个操作
 - 不同事务的冲突操作

冲突可串行化（续）

[例11.3] 今有调度

$$Sc_1 = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$$



$$Sc_2 = r_1(A)w_1(A)r_1(B)w_1(B)r_2(A)w_2(A)r_2(B)w_2(B)$$

T_1

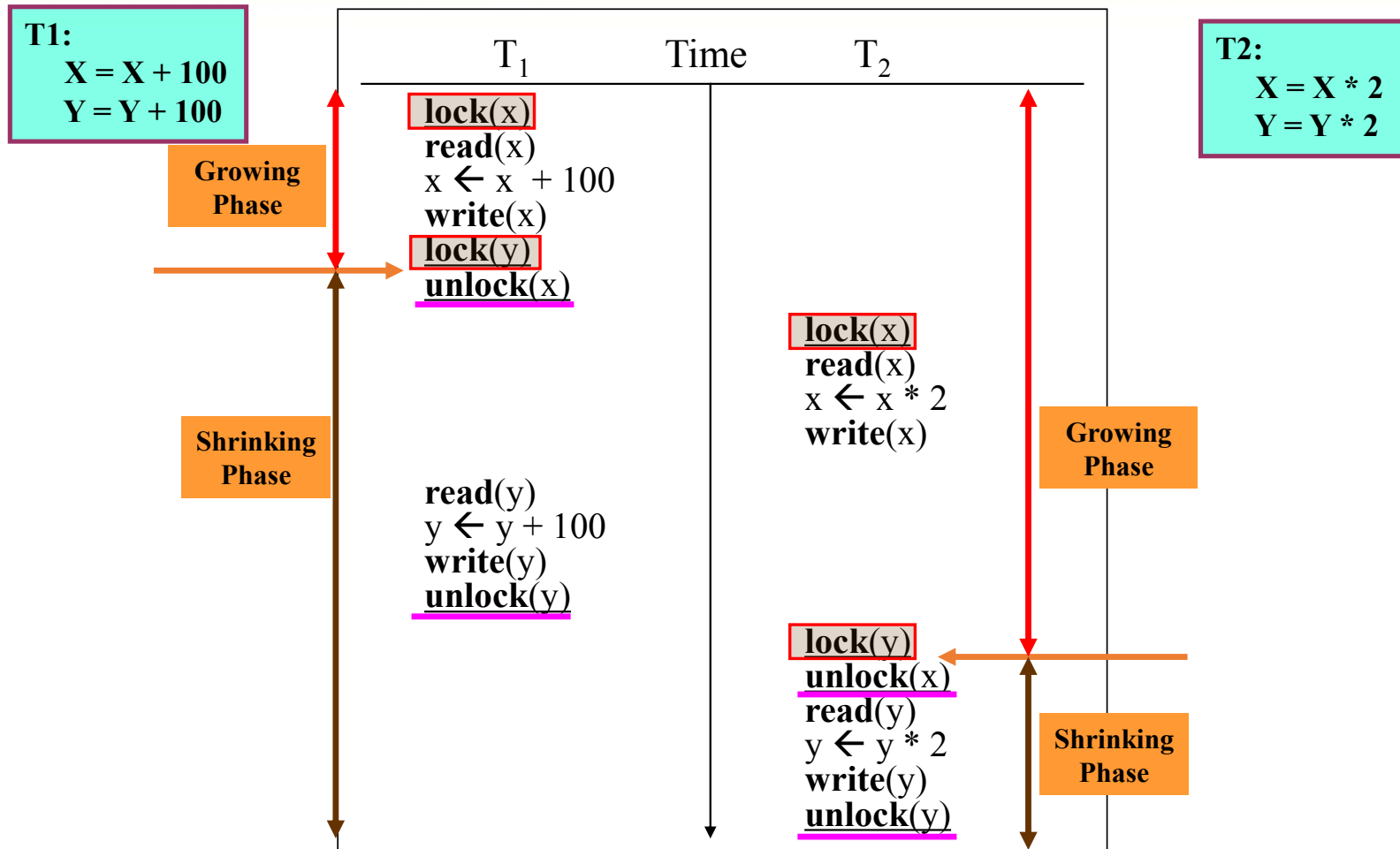
T_2

Sc_2 等价于一个串行调度 T_1, T_2 。所以 Sc_1 冲突可串行化的调度

5. 两段锁协议

- 数据库管理系统普遍采用两段锁协议的方法实现并发调度的可串行性，从而保证调度的正确性
- 两段锁协议，事务分为两个阶段
 - 第一阶段是获得封锁，也称为扩展阶段 **Growing Phase**
 - 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 **Shrinking Phase**
 - 事务可以释放任何数据项上的任何类型的锁，但是不能再申请任何锁

两段锁协议示例



6.封锁粒度

□封锁对象的大小称为封锁粒度(Granularity)

□封锁的对象:逻辑单元，物理单元

例：在关系数据库中，封锁对象：

■逻辑单元: 属性值、属性值的集合、元组、关系、索引项、整个索引、整个数据库等

■物理单元: 页（数据页或索引页）、物理记录等

选择封锁粒度原则

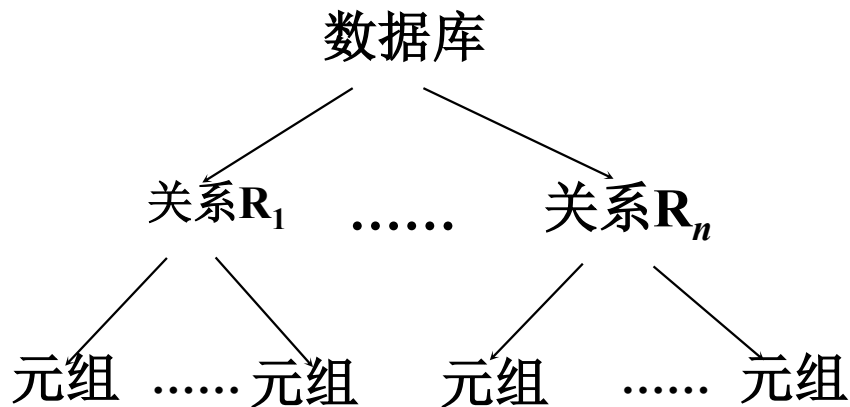
□封锁粒度与系统的并发度和并发控制的开销密切相关。

■封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；

■封锁的粒度越小，并发度较高，但系统开销也就越大

多粒度封锁（续）

例：三级粒度树。根结点为数据库，数据库的子结点为关系，关系的子结点为元组。



三级粒度树

意向锁

□ 引进意向锁（intention lock）目的

■ 提高对某个数据对象加锁时系统的检查效率

□ 如果对一个结点加意向锁，则说明该结点的下层结点正在被加锁

□ 对任一结点加基本锁，必须先对它的上层结点加意向锁

□ 例如，对任一元组加锁时，必须先对它所在的数据库和关系加意向锁

常用意向锁

- 意向共享锁(Intent Share Lock, 简称IS锁)
- 意向排它锁(Intent Exclusive Lock, 简称IX锁)
- 共享意向排它锁(Share Intent Exclusive Lock, 简称SIX锁)

意向锁（续）

□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：事务 T_1 要对 R_1 中某个元组加S锁，则要首先对关系 R_1 和数据库加IS锁

意向锁（续）

□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：事务 T_1 要对 R_1 中某个元组加X锁，则要首先对关系 R_1 和数据库加IX锁

意向锁（续）

□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 $SIX = S + IX$ 。

例：对某个表加SIX锁，则表示该事务要读整个表（所以要对该表加S锁），同时会更新个别元组（所以要对该表加IX锁）。

意向锁（续）

意向锁的相容矩阵

$T_1 \backslash T_2$	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

Y=Yes, 表示相容的请求

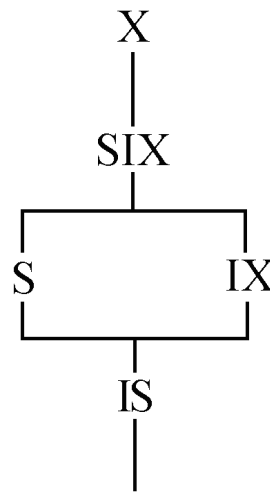
N=No, 表示不相容的请求

(a) 数据锁的相容矩阵

意向锁（续）

□ 锁的强度

- 锁的强度是指它对其他锁的排斥程度
- 一个事务在申请封锁时以强锁代替弱锁是安全的，反之则不然



(b) 锁的强度的偏序关系

意向锁（续）

□ 具有意向锁的多粒度封锁方法

- 申请封锁时应该按自上而下的次序进行
- 释放封锁时则应该按自下而上的次序进行

例如：事务 T_1 要对关系 R_1 加S锁

- 要首先对数据库加IS锁
- 检查数据库和 R_1 是否已加了不相容的锁(X或IX)
- 不再需要搜索和检查 R_1 中的元组是否加了不相容的锁(X锁)

练习

□课后练习题9

9. 设 **T1,T2,T3** 是的三个事务:

T1: A: =A+2;

T2: A: =A*2;

T3: A: =A**2; ($A \leftarrow A^2$)

设 A 的初值为 0;

- (1) 若这三个事务允许并发执行, 则有多少种可能的正确结果, 请一一列举出来;
- (2) 请给出一个可串行化的调度, 并给出执行结果;
- (3) 请给出一个非串行化的调度, 并给出执行结果;
- (4) 若这三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;
- (5) 若这三个事务都遵守两段锁协议, 请给出一个产生死锁的调度。

解: (1) 4 种 $A=16,8,4,2$

T1-T2-T3 $A=16$

T1-T3-T2 $A=8$

T2-T1-T3 或 T3-T1-T2 $A=4$

T2-T3-T1 或 T3-T2-T1 $A=2$

练习

□课后练习题9

(2) 一个可串行化的调度及执行结果如下图所示：

时间	T1	T2	T3
t1	Slock A		
t2	X=A=0		
t3	Unlock A		
t4	Xlock A		
t5		Slock A	
t6	A=X+2	等待	
t7	写回 A (=2)	等待	
t8	Unlock A	等待	
t9		获得 Slock A	
t10		X=A=2	
t11		Unlock A	
t12		Xlock A	
t13			Slock A
t14		A=X*2	等待
t15		写回 A (=4)	等待
t16		Unlock A	等待
t17			获得 Slock A
t18			X=A=4
t19			Unlock A
t20			Xlock A
t21			A=X ²
t22			写回 A (=16)
t23			Unlock A

执行结果为 A=16，是可串行化的调度。

练习

□课后练习题9

(3) 一个非串行化调度及执行结果如下图所示：

时间	T1	T2	T3
t1	Slock A		
t2	X=A=0		
t3	Unlock A		
t4		Slock A	
t5		X=A=0	
t6	Xlock A		
t7	等待	Unlock A	
t8	获得 Xlock A		
t9	A=X+2		
t10	写回 A (=2)		
t11	Unlock A		Slock A
t12			等待
t13			获得 Slock A
t14			X=A=4
t15			Unlock A
t16		Xlock A	Xlock A
t17		等待	
t18		等待	A=X ²
t19		等待	写回 A (=4)
t20		获得 Xock A	Unlock A
t21		A=X*2	
t22		写回 A (=0)	
t23		Unlock A	

运行结果 A=0，为非串行化调度。

练习

□课后练习题9

(4) 若三个串行事务都遵守两段锁协议，下图是按 T3—T1—T2 顺序运行的一个不产生死锁的可串行化调度；

时间	T1	T2	T3
t1			Slock A
t2			X=A=0
t3			Xlock A
t4	Slock A		A=X ²
t5	等待		写回 A (=0)
t6	等待		Unlock A
t7	X=A=0		
t8	Xlock A		
t9	等待	Slock A	Unlock A
t10	A=X+2	等待	
t11	写回 A (=2)	等待	
t12	Unlock A	等待	
t13		X=A=2	
		Xlock A	
		等待	
		A=X*2	
		写回 A (=4)	
		Unlock A	
	Unlock A	Unlock A	

从上可见，按照 T3—T1—T2 的顺序执行结果 A=4 完全与串行化调度相同，所以是一个不产生死锁的可串行化的调度。

练习

□课后练习题9

(5) 若三个事务都遵守两段锁协议，下图是一个产生死锁的调度。

时间	T1	T2	T3
t1	Slock A		
t2	X=A=0		
t3		Slock A	
t4		X=A=0	
t5	Xlock A		
t6	等待		
t7		Xlock A	
t8		等待	
			Slock A X=A=0 Xlock A 等待

上例中，T1 申请对 X1 加写锁，由于 T2 对 X1 加了读锁，所以不成功，处于等待状态；T2 申请对 A 加写锁，由于 T1 对 A 加了读锁，所以不成功，处于等待状态；T3 申请对 A 加读锁，由于 T1 对 A 加了读锁，所以不成功，处于等待状态。因此，三个事务都处于等待状态，产生死锁。