CONTENTS

并发控制概述

- 并发控制机制大体上可分为悲观的和乐观的两种。
- 悲观的并发控制方法认为数据库的一致性经常会受到破坏，因此在事务访问数据对象前须采取一定措施加以控制，只有得到访问许可时，才能访问数据对象，如基于封锁的并发控制方法。——事前控制
- 乐观的并发控制方法则认为数据库的一致性通常不会遭到破坏，故事务执行时可直接访问数据对象，只在事务结束时才验证数据库的一致性是否会遭到破坏，如基于有效性验证方法。——事后验证
- 本章介绍基于封锁的并发控制方法。

基于封锁方法的基本思想

■基本思想：当事务 T 需访问数据对象 Q 时，先申请对 Q 的锁。如批准获得，则事务 T 继续执行，且此后不允许其他任何事务修改 Q，直到事务 T 释放 Q 上的锁为止。

■基本锁类型：

- 共享锁（shared lock，记为S）：如果事务 T 获得了数据对象 Q 的共享锁，则事务 T 可读 Q 但不能写 Q。
- 排它锁（eXclusive lock，记为X）：如果事务 T 获得了数据对象 Q 上的排它锁，则事务 T 既可读 Q 又可写 Q。

锁相容性

■ “锁相容”是指如果 T_i 已持有数据对象 Q 的某类型锁后，事务 T_j 也申请对 Q 的封锁。如果允许事务 T_j 获得对 Q 的锁，则称事务 T_j 申请锁类型与事务 T_i 的持有锁类型相容；否则称为不相容。

■基本锁类型的封锁相容性原则：

- 共享锁与共享锁相容
- 排它锁与共享锁、排它锁与排它锁是不相容的。

T_j \ T_i	S	X
S	+	—
X	—	—

“+”表示相容

“—”表示不相容

图 -12　基本锁类型的相容性矩阵

事务封锁举例

单个事务封锁举例

■申请和释放锁操作：

- SL(*Q*)—— 申请数据对象 *Q* 上的共享锁；
- XL(*Q*)—— 申请数据对象 *Q* 上的排它锁；
- UL(*Q*)—— 释放数据对象 *Q* 上的锁。

封锁能否保证并发执行事务的冲突可串行化?

T_4	T_5	A	B
R(A)		10	
A=A-2			
W(A)		8	
	R(A)	8	
	A=A-3		
	W(A)	5	
R(B)			15
B=B-2			
W(B)			13
	R(B)		13
	B=B-3		
	W(B)		10

图-6 T_4和T_5并发执行

T_4

XL(A)
R(A)
A=A-2
W(A)
UL(A)
XL(B)
R(B)
B=B-2
W(B)
UL(B)
COMMIT

其他事务在此期间不允许访问数据对象 A

其他事务在此期间不允许访问数据对象 B

图 -13　增加了封锁的事务 T_4 操作序列

并发事务封锁举例

■[例　7] 考虑并发事务 T_1、 T_2和 T_3，它们申请锁和释放锁的规则是：

- 访问数据对象前根据操作类型申请锁；
- 访问完后立即释放锁；
- 当一个事务释放锁后，由等待时间较长的事务优先获得锁。

它们的一个可能的并发执行过程如图 -14 所示。

并发事务封锁举例

步骤	T_1	T_2	T_3
1			SL(*A*)
2	XL(*A*)		
3	等待	XL(*A*)	
4	等待	等待	R(*A*)
5	等待	等待	UL(*A*)
6	R(*A*)	等待	
7		等待	SL(*A*)
8	*A*=*A*-2	等待	等待
9	W(*A*)	等待	等待
10	UL(*A*)	等待	等待
11		R(*A*)	等待
12	ROLLBACK		等待
13		*A*=*A*-3	等待
14		W(*A*)	等待
15		UL(*A*)	等待
16		COMMIT	R(*A*)
17			UL(*A*)
18			COMMIT

图 -14　T_1、T_2 和 T_3 上锁操作序列

并发事务封锁举例

该调度避免了丢失更新，即不会有多个写事务读取同一数据对象的相同值，因为一个数据对象任何时候只能有一个排它锁。

步骤	T_1	T_2	T_3
1			SL(*A*)
2	XL(*A*)		
3	等待	XL(*A*)	
4	等待	等待	R(*A*)
5	等待	等待	UL(*A*)
6	R(*A*)	等待	
7		等待	SL(*A*)
8	*A*=*A*-2	等待	等待
9	W(*A*)	等待	等待
10	UL(*A*)	等待	等待
11		R(*A*)	等待
12	ROLLBACK		等待
13		*A*=*A*-3	等待
14		W(*A*)	等待
15		UL(*A*)	等待
16		COMMIT	R(*A*)
17			UL(*A*)
18			COMMIT

图 -14　T_1、T_2 和 T_3 上锁操作序列

并发事务封锁举例

- 但仍然存在以下问题：
  - 读脏数据。如 T_2 在步骤 11 读了 T_1 修改后的数据，而 T_1 在步骤 12 需 ROLLBACK 。

步骤	T_1	T_2	T_3
1			SL(*A*)
2	XL(*A*)		
3	等待	XL(*A*)	
4	等待	等待	R(*A*)
5	等待	等待	UL(*A*)
6	R(*A*)	等待	
7		等待	SL(*A*)
8	*A*=*A*-2	等待	等待
9	W(*A*)	等待	等待
10	UL(*A*)	等待	等待
11		R(*A*)	等待
12	ROLLBACK		等待
13		*A*=*A*-3	等待
14		W(*A*)	等待
15		UL(*A*)	等待
16		COMMIT	R(*A*)
17			UL(*A*)
18			COMMIT

图 -14　T_1 、 T_2 和 T_3 上锁操作序列

并发事务封锁举例

- 但仍然存在以下问题:
  - 不可重复读。如 T_3 两次读到 A 的值不同。

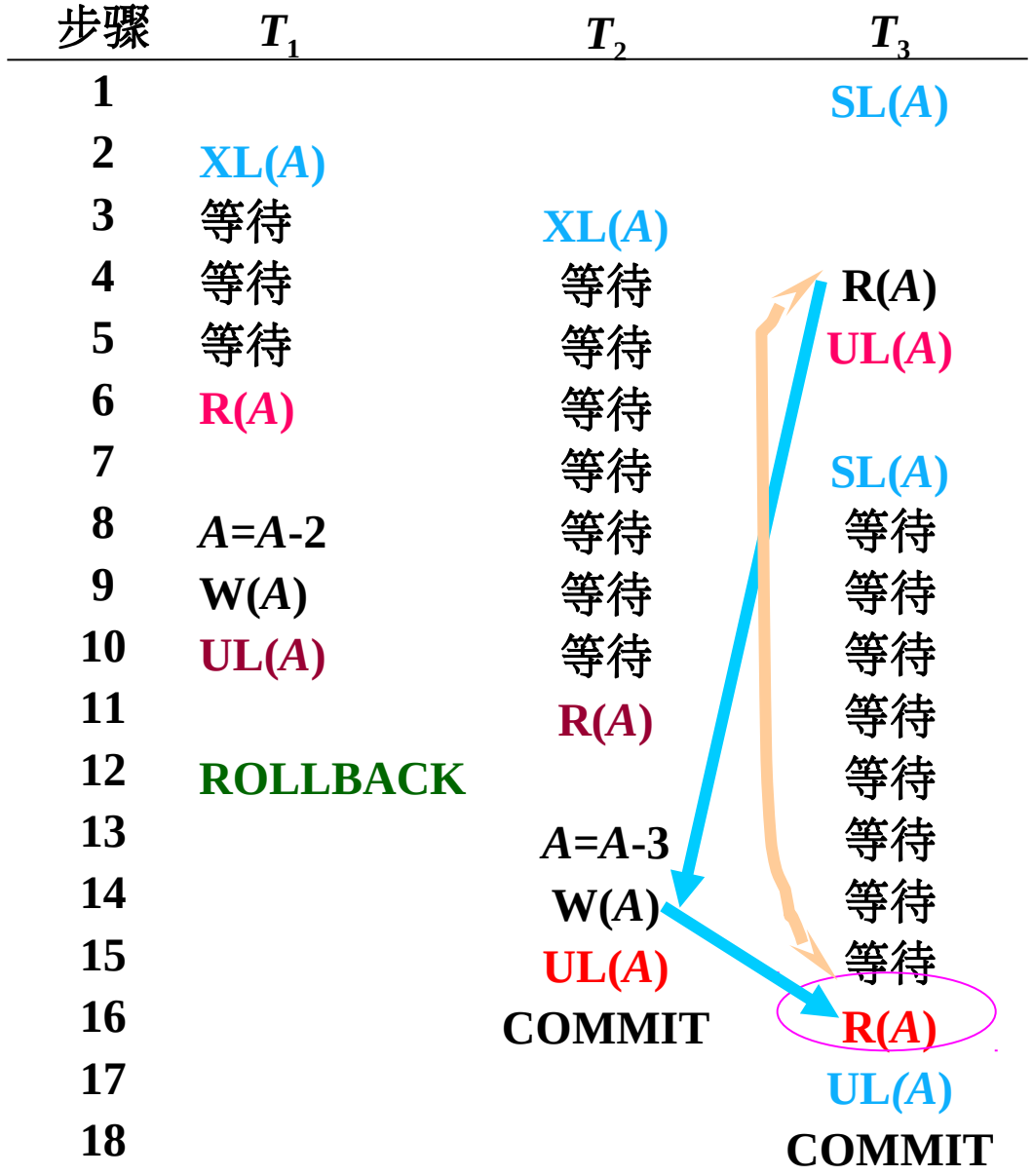

图 -14 T_1、T_2 和 T_3 上锁操作序列

并发事务封锁举例

- 但仍然存在以下问题：
  - 不可串行化。无论如何交换非冲突操作，上述调度都不能等价于 T_1、T_2 和 T_3 的任何一个串行调度。

出现上述问题的原因是事务过早释放了其持有的锁！

步骤	T_1	T_2	T_3
1			SL(*A*)
2	XL(*A*)		
3	等待	XL(*A*)	
4	等待	等待	R(*A*)
5	等待	等待	UL(*A*)
6	R(*A*)	等待	
7		等待	SL(*A*)
8	*A*=*A*-2	等待	等待
9	W(*A*)	等待	等待
10	UL(*A*)	等待	等待
11		R(*A*)	等待
12	ROLLBACK		等待
13		*A*=*A*-3	等待
14		W(*A*)	等待
15		UL(*A*)	等待
16		COMMIT	R(*A*)
17			UL(*A*)
18			COMMIT

图 -14　T_1、T_2 和 T_3 上锁操作序列

两阶段封锁协议

■两阶段封锁协议要求每个事务分两个阶段完成封锁操作：增长（申请锁）阶段和缩减（释放锁）阶段：

- 增长阶段：事务可以获得锁，但不能释放锁；
- 缩减阶段：事务可以释放锁，但不能获得新锁。

■定理：若所有事务均遵从两段锁协议，则这些事务的所有并行调度都是可串行化的。

■对于任何事务，调度中该事务获得其最后加锁的时刻（增长阶段结束点）称为事务的封锁点。这样，多个事务可以根据它们的封锁点进行排序，而这个顺序就是并发事务的一个冲突可串行化顺序。

两阶段封锁协议举例

[例 8] 图-15 采用了两阶段封锁，允许事务 T_4 在获得全部锁后（A 和 B 上的排它锁）提前释放部分锁（如步骤 7 释放了 A 上的排它锁），事务 T_5 得以提前执行，从而提高了事务 T_4 和 T_5 的并发度。

冲突可串行化，它等价于 $<T_4,T_5>$ 串行调度。

步骤	T_4	T_5
1	XL(A)	
2		XL(*A*)
3	R(A)	等待
4	A=A-2	等待
5	W(A)	等待
6	XL(B)	等待
7	UL(A)	等待
8		R(*A*)
9		A=*A*-3
10	R(B)	
11	B=B-2	
12		W(*A*)
13		XL(*B*)
14	W(B)	等待
15	UL(B)	等待
16		UL(*A*)
17		R(*B*)
18		*B*=*B*-3
19		W(*B*)
20		UL(*B*)

图 -15 T_4 和 T_5 的两阶段封锁

两阶段封锁协议存在的问题

■问题一： 可能导致死锁

- 持有锁事务出现相互等待都不能继续执行。

T_4	T_6
XL(A)	
R(A)	
	XL(B)
	R(B)
$A=A-2$	
W(A)	
	$B=B-3$
	W(B)
XL(B)	
(等待 T_6 释放 B 上的排它锁)	
	XL(A)
	(等待 T_4 释放 A 上的排它锁)

图 -16　采用两阶段封锁时 T_4 和 T_6 出现死锁

两阶段封锁协议存在的问题

■问题二：不能避免读脏数据

（T_2 读取了 T_1 的未提交更新结果）

导致的后果是“级联回滚”！

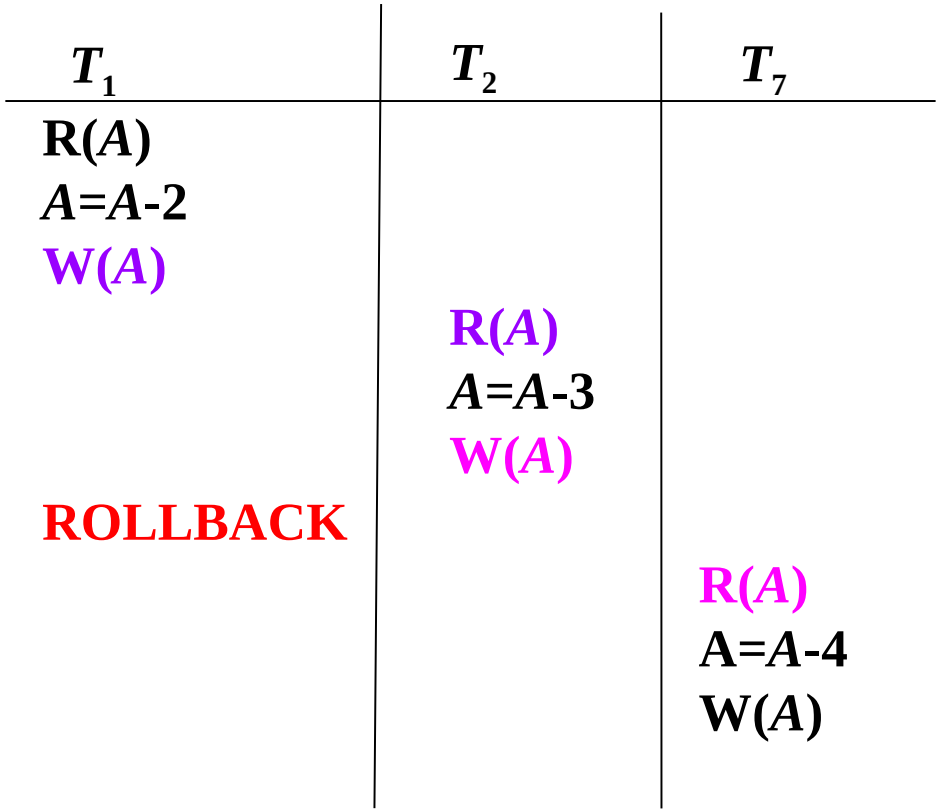

图 -17　由于读脏数据引起的级联回滚

两阶段封锁协议变体

- 对于级联回滚可以通过将两阶段封锁修改为严格两阶段封锁协议加以避免。
- 严格两阶段封锁协议除了要求封锁是两阶段之外，还要求事务持有的所有排它锁必须在事务提交后方可释放。这个要求保证了未提交事务所写的任何数据在该事务提交之前均以排它方式加锁，防止了其他事务读取这些数据。
- 另一个两阶段封锁的变体是强两阶段封锁协议，它要求事务提交之前不得释放任何锁（包括共享锁和排它锁）。

小结

- 锁的作用
- 两阶段封锁协议