

第十一章 并发控制

11.1 并发控制概述

11.2 封锁

11.3 封锁协议

11.4 活锁和死锁

11.5 并发调度的可串行性

11.6 两段锁协议

11.7 封锁的粒度

*11.8 其他并发控制机制

11.9 小结





11.2 封锁

- ❖ 什么是封锁
- ❖ 基本封锁类型
- ❖ 锁的相容矩阵



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。

T_1	T_2
①  $R(A)=16$	
②	$R(A)=16$
③ $A \leftarrow A-1$	
 $W(A)=15$	
④	$A \leftarrow A-3$
	$W(A)=13$

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。
- ❖ 封锁是实现并发控制的一个非常重要的技术



基本封锁类型

❖ 一个事务对某个数据对象加锁后究竟拥有什么样的控制由封锁的类型决定。

❖ 基本封锁类型

- 排它锁（**Exclusive Locks**，简记为**X锁**）

- 共享锁（**Share Locks**，简记为**S锁**）



排它锁

- ❖ 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
- ❖ 排它锁又称为写锁



共享锁

- ❖ 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
- ❖ 共享锁又称为读锁



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

T_2	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

T_2	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



锁的相容矩阵

$T_1 \backslash T_2$	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求



第十一章 并发控制

11.1 并发控制概述

11.2 封锁

11.3 封锁协议

11.4 活锁和死锁

11.5 并发调度的可串行性

11.6 两段锁协议

11.7 封锁的粒度

*11.8 其他并发控制机制

11.9 小结



11.3 封锁协议

❖ 什么是封锁协议

- 在运用**X**锁和**S**锁对数据对象加锁时，需要约定一些规则，这些规则为封锁协议（**Locking Protocol**）。
 - 何时申请**X**锁或**S**锁
 - 持锁时间
 - 何时释放
- 对封锁方式规定不同的规则，就形成了各种不同的封锁协议，在不同的程度上保证并发操作的正确调度。



保持数据一致性的常用封锁协议

❖ 三级封锁协议

1. 一级封锁协议

2. 二级封锁协议

3. 三级封锁协议



1. 一级封锁协议

❖ 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。

- 正常结束（**COMMIT**）

- 非正常结束（**ROLLBACK**）

❖ 一级封锁协议可防止丢失修改，并保证事务T是可恢复的。



使用封锁机制解决丢失修改问题

例:

T ₁	T ₂
① <u>R(A)=16</u>	
②	<u>R(A)=16</u>
③ <u>A←A-1</u> <u>W(A)=15</u>	
④	<u>A←A-3</u> <u>W(A)=13</u>

T ₁	T ₂
① <u>Xlock A</u>	
② R(A)=16	
	<u>Xlock A</u>
③ A←A-1	等待
W(A)=15	等待
<u>Commit</u>	等待
<u>Unlock A</u>	等待
④	获得Xlock A
	<u>R(A)=15</u>
	<u>A←A-3</u>
⑤	<u>W(A)=12</u>
	Commit
没有丢失修改	Unlock A

一级封锁协议（续）

- ❖ 在一级封锁协议中，如果仅仅是读数据不对其进行修改，是不需要加锁的，所以它不能保证可重复读和不读“脏”数据。



使用一级封锁协议不能解决的问题

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

不可重复读

使用一级封锁协议不能解决的问题

T ₁	T ₂
① R(C)=100 C←C*2 W(C)=200	
②	R(C)=200
③ ROLLBACK C恢复为100	

读“脏”数据

T ₁	T ₂
① Xlock C 获得	
② R(C)=100 C←C*2 W(C)=200	
③	R(C)=200
④ Rollback C恢复为100 Unlock C	

读“脏”数据

2. 二级封锁协议

❖ 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。

❖ 二级封锁协议可以防止丢失修改和读“脏”数据。



使用二级封锁协议解决丢失修改问题

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
	$A \leftarrow A-3$
⑤	W(A)=12
	Commit
	Unlock A

没有丢失修改

- 事务T1在读A进行修改之前先对A加X锁
- 当T2再请求对A加X锁时被拒绝
- T2只能等待T1释放A上的锁后T2获得对A的X锁
- 这时T2读到的A已经是T1更新过的值15
- T2按此新的A值进行运算, 并将结果值A=14送回到磁盘。避免了丢失T1的更新。



使用封锁机制解决读“脏”数据问题

例

T ₁	T ₂
① R(C)=100 C←C*2 W(C)=200	
②	R(C)=200
③ ROLLBACK C恢复为100	

读“脏”数据

T ₁	T ₂
① Xlock C R(C)=100 C←C*2 W(C)=200	
②	<u>Slock C</u> 等待
③ ROLLBACK (C恢复为100) Unlock C	等待 等待 等待
④	<u>获得Slock C</u> R(C)=100 Commit C Unlock C

未读“脏”数据

二级封锁协议（续）

❖ 在二级封锁协议中，由于读完数据后即可释放S锁，所以它不能保证可重复读。



使用二级封锁协议不能解决的问题: 不可重复读

T ₁	T ₂
① R(A)=50 R(B)=100 求和=150	
②	R(B)=100 B←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	T ₂
① Slock A 获得 读A=50 Unlock A	
② Slock B 获得	
③	Xlock B 等待 等待
④ 读B=100 Unlock B 求和=150	
⑤	获得 读B=100 B←B*2 写回B=200 Commit Unlock B

T ₁ (续)	T ₂
⑥ Slock A 获得 读A=50 Unlock A Slock B 获得 读B=200 Unlock B 求和=250 (验算不对)	

不可重复读

3. 三级封锁协议

❖ 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。

❖ 三级封锁协议可防止丢失修改、读脏数据和不可重复读。



使用封锁机制解决不可重复读问题

T ₁	T ₂
① <u>Sclock A</u> 获得 读A=50 <u>Unlock A</u>	
② <u>Sclock B</u> 获得	
③	<u>Xlock B</u>
④ <u>读B=100</u> <u>Unlock B</u> 求和=150	等待 等待
⑤	获得 <u>读B=100</u> <u>B ← B * 2</u> <u>写回B=200</u> Commit <u>Unlock B</u>

T ₁ (续)	T ₂
⑥ <u>Sclock A</u> 获得 读A=50 <u>Unlock A</u> <u>Sclock B</u> 获得 读B=200 <u>Unlock B</u> 求和=250 <u>(验算不对)</u>	

不可重复读

T ₁	T ₂
① <u>Slock A</u> <u>Slock B</u> R(A)=50 R(B)=100 <u>求和=150</u>	
②	<u>Xlock B</u> 等待 等待 等待 等待 等待 等待 等待
③ R(A)=50 R(B)=100 <u>求和=150</u> Commit <u>Unlock A</u> <u>Unlock B</u>	
④	获得XlockB R(B)=100 B ← B * 2 W(B)=200 Commit Unlock B
⑤	

可重复读

4. 封锁协议小结

	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		√		√	√	√	√

不同级别的封锁协议和一致性保证

❖ 三级协议的主要区别

- 什么操作需要申请封锁以及何时释放锁（即持锁时间）

❖ 不同的封锁协议使事务达到的一致性级别不同

- 封锁协议级别越高，一致性程度越高



小结

❖ 基本封锁类型

- X锁
- S锁

❖ 封锁协议

- 一级封锁协议
- 二级封锁协议
- 三级封锁协议



思考题

- ❖ 在三级封锁协议中，写锁都是长锁，在事务结束时释放。请思考，如果在一级封锁协议中，将写锁改为短锁，是否能防止丢失修改？如果能，请说明理由。如果不能，请给出实例。



