Week16_Course

Database System
Concurrency Control part4_1

Shared locks Protocol

Concurrency Control

- Schedule
- Conflict Serializable Schedule
- Two phase locking
- Warning Protocol
- Validation

Database System - Nankai

- Beyond this simple 2PL protocol, it is all a matter of improving performance and allowing more concurrency….
  - Shared locks
  - Multiple granularity
  - Inserts, deletes and phantoms
  - Other types of C.C. mechanisms
    - Timestamping
    - Validation

Shared locks

So far only exclusive locks:

$S = \ldots l_1(A)\ r_1(A)\ u_1(A) \cdots l_2(A)\ r_2(A)\ u_2(A) \cdots$

Do not conflict –> locking unnecessary

Instead, use **shared** locks (S) for reading:

S=... ls1(A) r1(A) ls2(A) r2(A) .... u1(A)u2(A)

Write actions conflict –> use **exclusive** (X) locks for writing

Lock actions:

$l\text{-}m_k(A)$: lock A in mode m (S or X) for T_k

$u_k(A)$: release (whatever) lock(s) held by transaction T_k on element A

Rule #1 Well-formed transactions

T_i =... I-$S_1(A)$ ⋯ $r_1(A)$ ⋯ $u_1(A)$ ⋯
T_i =... I-$X_1(A)$ ⋯ $w_1(A)$ ⋯ $u_1(A)$ ⋯

- Request
  - an S-lock for reading
  - an X-lock for writing
- Release the locks eventually

- What about transactions that first read and later write the same element?

Option 1: Request exclusive lock

T_i = ...l-X_1(A) ··· r_1(A) ... w_1(A) ... u_1(A) ···

Option 2: Upgrade

(E.g., need to read, but don't know if will write•••)

Ti=... l-S1(A) ••• r1(A) ... l-X1(A) •••w1(A) ...u(A)•••

Tj= ... l-S2(A) ... r2(A)

Think s getting 2nd lock on A

Rule #2 Legal scheduler

$S = ...l\text{-}S_i(A) \ldots \quad \ldots u_i(A) \ldots$

$\longleftrightarrow$

no $l\text{-}X_j(A)$ for $j \neq i$

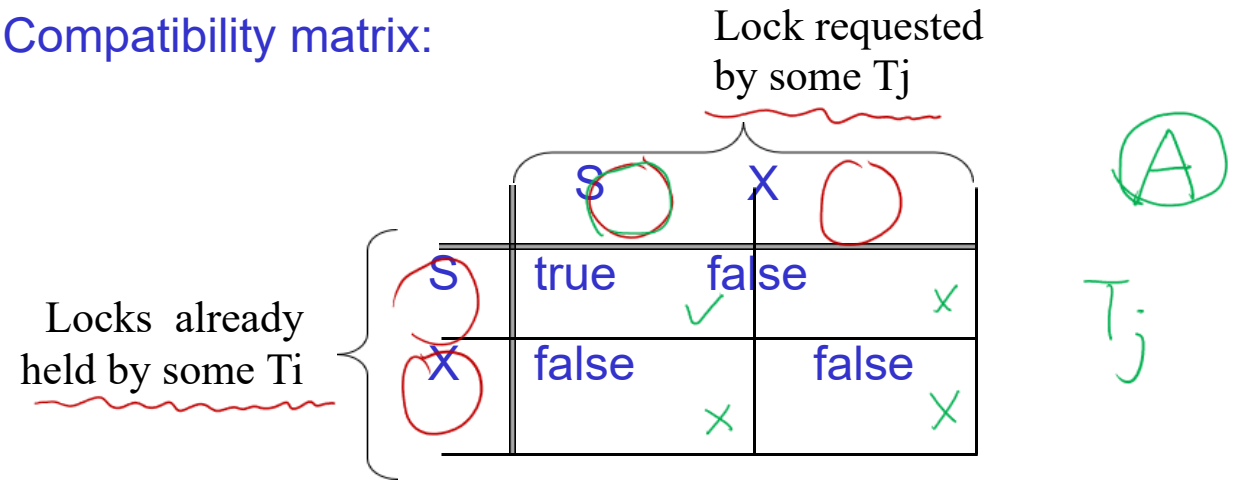
$S = \ldots l\text{-}X_i(A) \ldots \quad \ldots u_i(A) \ldots$

$\longleftrightarrow$

no $l\text{-}X_j(A)$ for $j \neq i$
no $l\text{-}S_j(A)$ for $j \neq i$

A way to summarize Rule #2

Compatibility matrix:



True <=> OK to give a new lock of requested kind

Rule # 3 (2PL)

Only change to previous:

Lock upgrades

$S(A) \rightarrow \{S(A), X(A)\}$ or

$S(A) \rightarrow X(A)$

are allowed only in the growing phase

Theorem Rules 1,2,3 ⇒ Conf.serializable for S/X locks schedules

Proof: Similar to the X locks case

Lock types beyond S/X

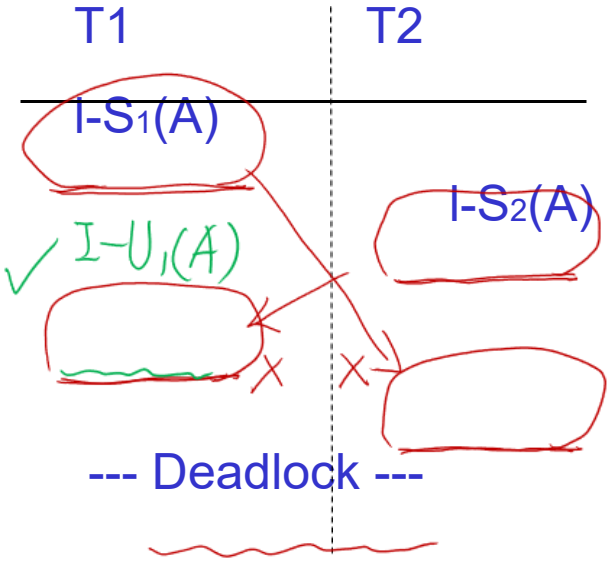
Examples:

(1) update lock

(2) increment lock (see the textbook)

Update locks

A common deadlock problem with upgrades:



Option 2: Upgrade

(E.g., need to read, but don't know if will write…)

Ti=… l-S1(A) … r1(A) … l-X1(A) …w1(A) …u(A)…

U A

Database System - Nankai

Solution

If T_i wants to read A and *knows* it may later want to write A, it requests an update lock (not shared) $l\text{-}u_k(A)$

New request

Comp

Lock already held in

	S	X	U	
S	T	F	F	T
X	F	F	F	F
U	F	F	F	F

As before

A common deadlock problem with update:

T1	T2
I-$u_1(A)$;$r_1(A)$	
	I-$u_2(A)$被拒绝
I-$X_1(A)$;$w_1(A)$;$u_1(A)$	
	I-$u_2(A)$;$r_2(A)$;
	I-$X_2(A)$;$w_2(A)$;$u_2(A)$

Note: object A may be locked in different modes at the same time...

S_1=...I-S_1(A)…I-S_2(A)…I-U_3(A)…

- To grant a lock in mode m, mode m must be compatible with all currently held locks on object

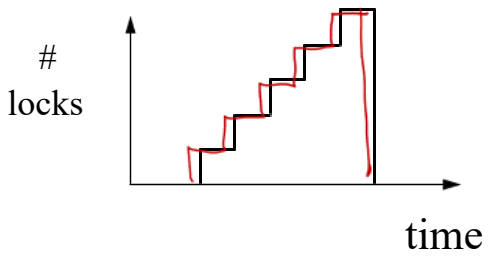
How does locking work in practice?

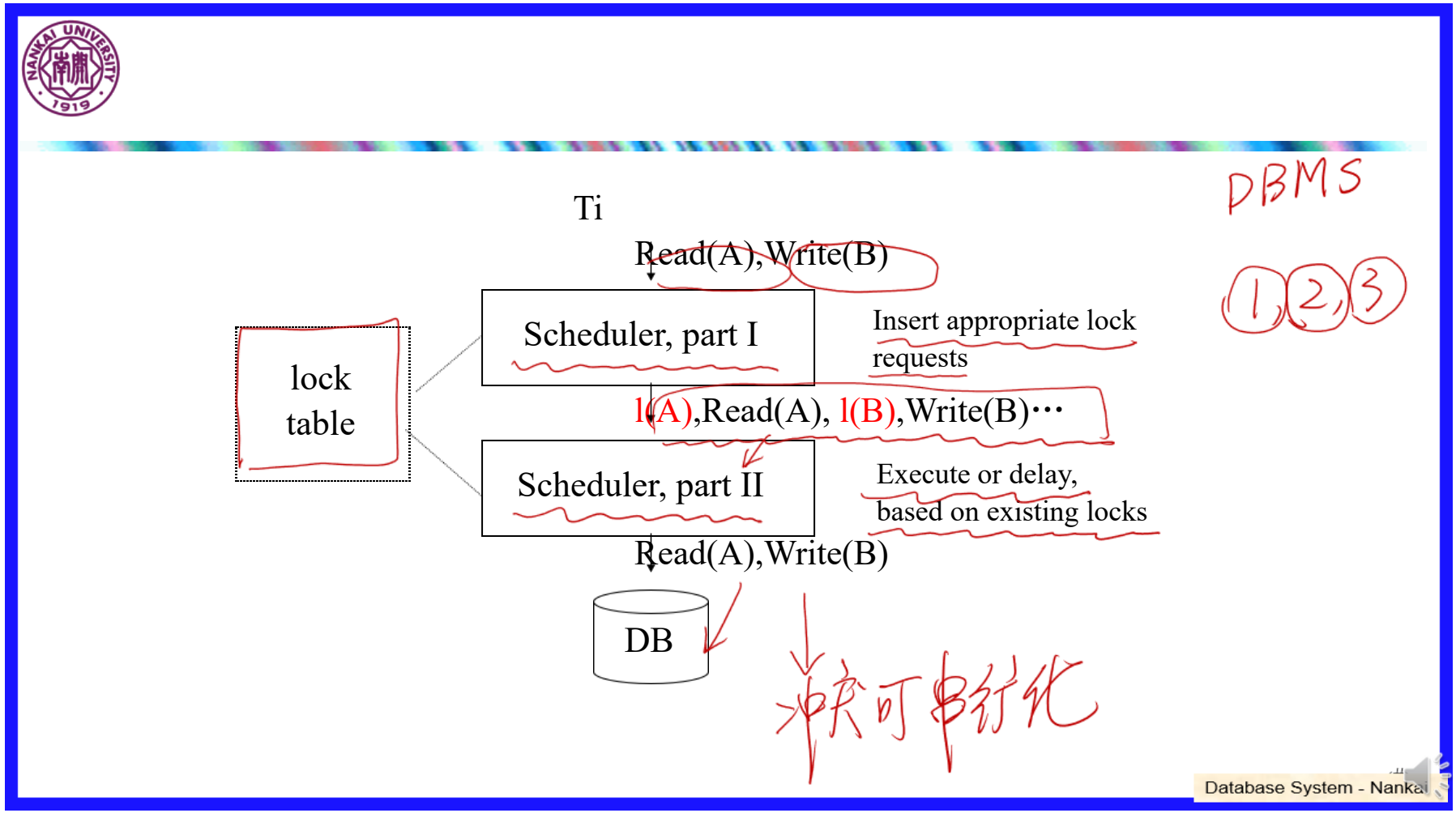
- Every system is different
  (E.g., may not even provide
  CONFLICT-SERIALIZABLE schedules)
- Here is one (simplified) way ...

Sample Locking System:

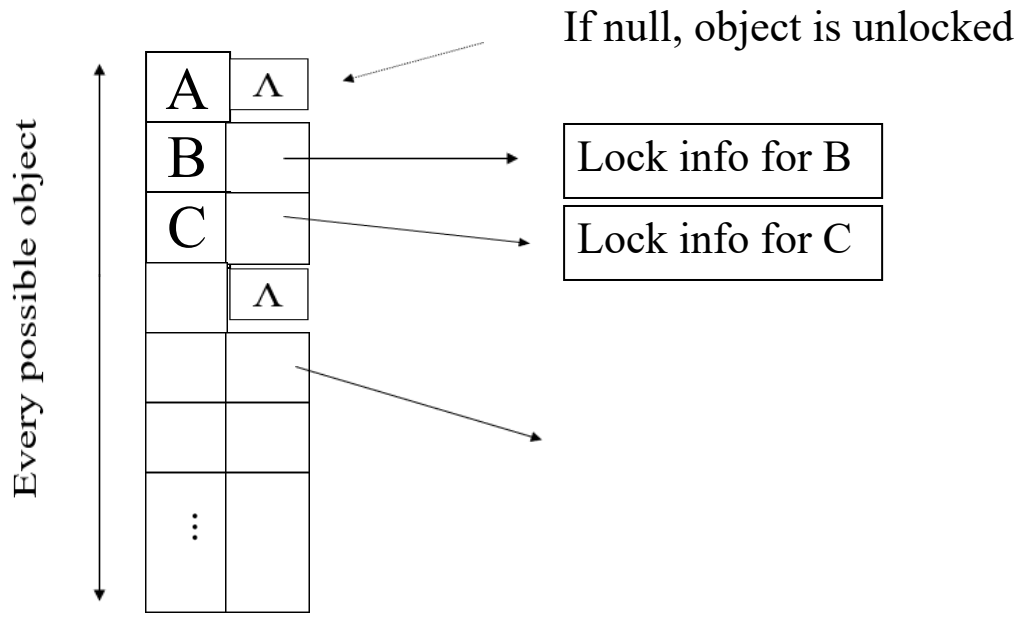
(1) Don't trust transactions to request/release locks

(2) Do not release locks until transaction commits/aborts:



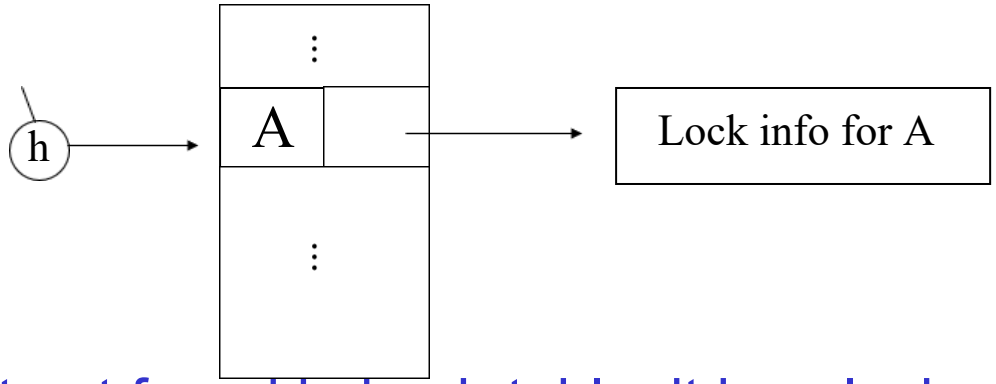

Ti
Read(A),Write(B)
Scheduler, part I
Insert appropriate lock requests
lock table
l(A),Read(A), l(B),Write(B)…
Scheduler, part II
Execute or delay, based on existing locks
Read(A),Write(B)
DB
DBMS
1,2,3
Database System - Nankai

Lock table Conceptually



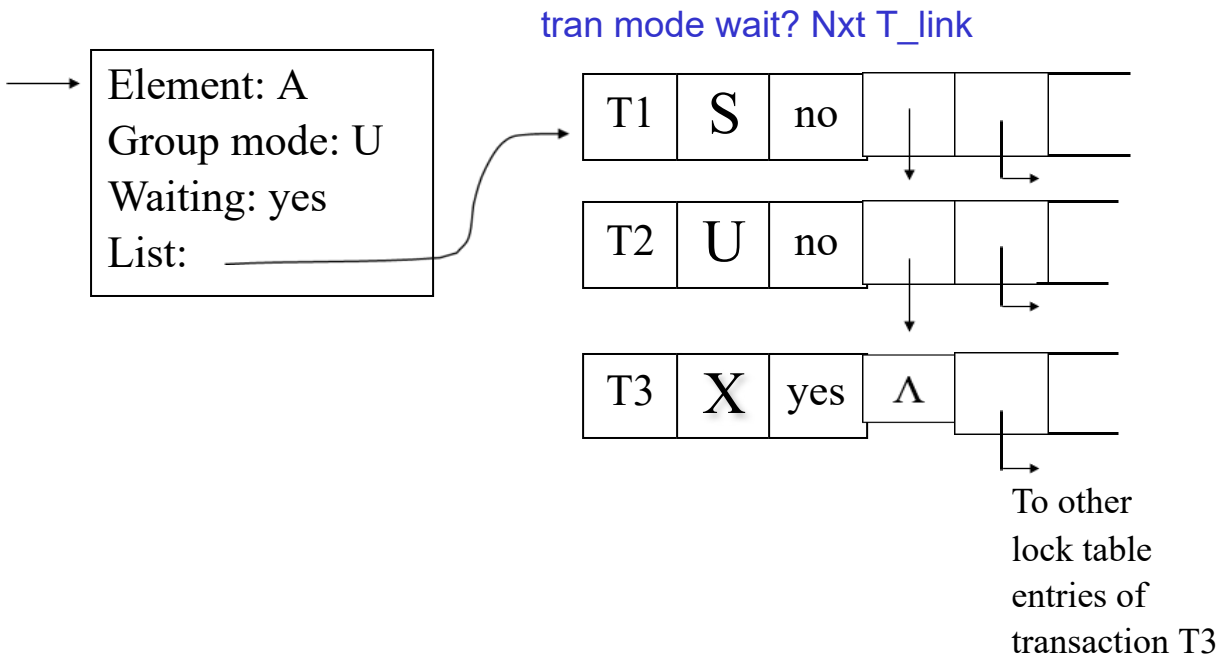
But use hash table:

A



If object not found in hash table, it is unlocked

Lock info for A - example



多选题 1分

互动交流一

以下哪几个事务是Well-formed transactions并且满足两段锁协议？

A T=ls(A) lx(B) r(A) w(B) u(B) lx(C) w(C) u(C)

B T=ls(A) r(A) lx(B) w(B) u(B) lx(C) w(C) u(C) u(A)

C T=ls(A) r(A) u(A) ls(B) w(B) u(B) lx(C) w(C) u(C)

D T=ls(A) lx(B) r(A) w(B) lx(C) u(B) w(C) u(C) u(A)

提交

se System - Nankai

多选题 1分

互动交流二 —不定项选择题

以下哪几个调度满足合法调度(Legal scheduler)的规则?

A $S = ls_1(A)\ r_1(A)\ ls_1(B)\ r_1(B)\ u_1(A)\ ls_2(B)\ r_2(B)\ u_2(B)\ ls_3(B)\ r_3(B)\ u_1(B$

B $S = ls_1(A)\ r_1(A)\ lx_1(B)\ w_1(B)\ ls_2(B)\ u_1(B)\ u_1(A)\ r_2(B)\ u_2(B)\ ls_3(B)\ r_3(B$

C $S = ls_1(A)\ r_1(A)\ ls_1(B)\ r_1(B)\ u_1(A)\ lx_2(B)\ w_2(B)\ u_2(B)\ ls_3(B)\ r_3(B)\ u_1(B$

提交

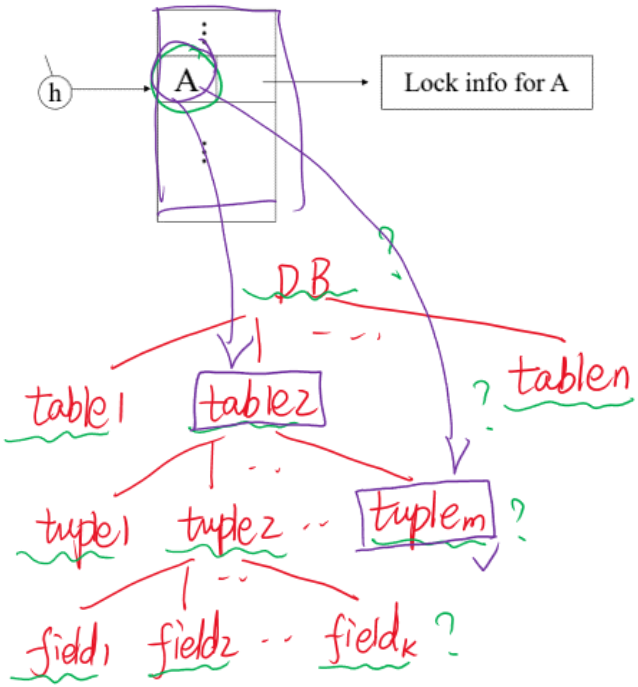
Database System - Nankai

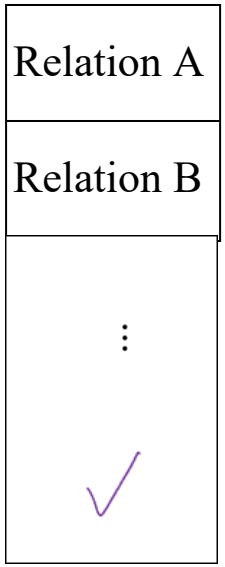
Week16_Course

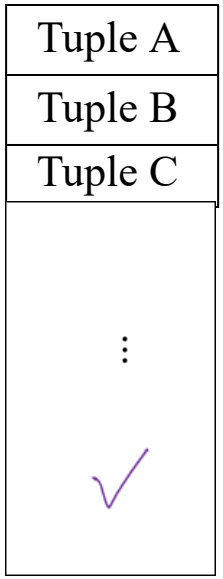
Database System
Concurrency Control part4_2

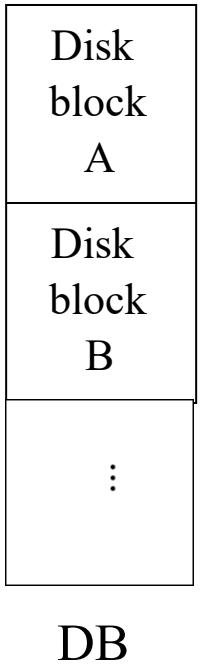
Warning locks Protocol

What are the objects we lock?







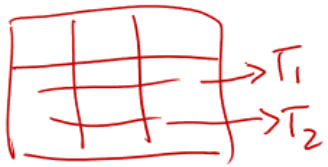


?

Database System - Nankai

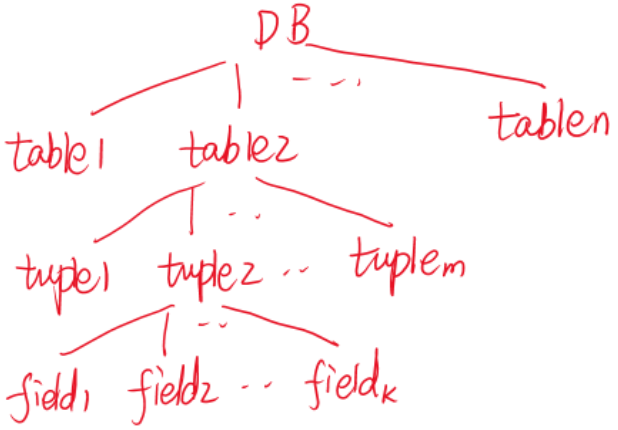
- Locking works in any case, but should we choose small or large objects?

- If we lock large objects (e.g., Relations)
  - Need few locks
  - Get low concurrency
- If we lock small objects (e.g., tuples,fields)
  - Need more locks (=> overhead higher)
  - Get more concurrency

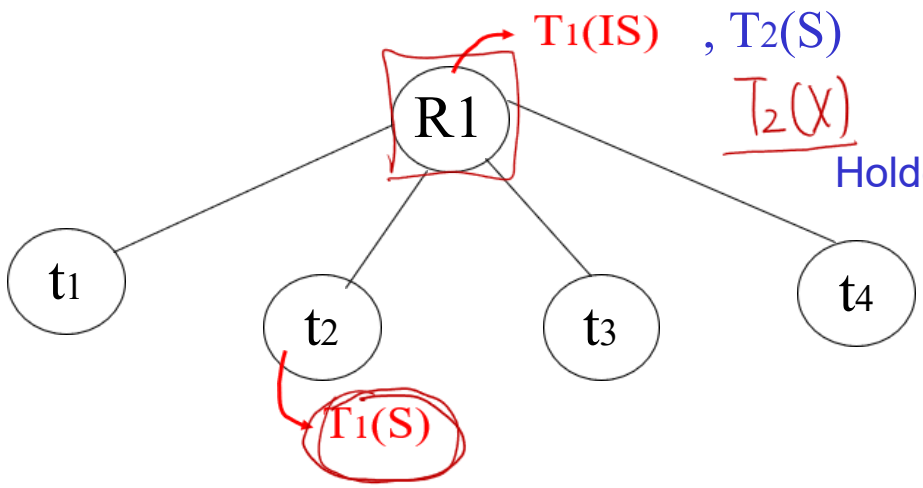


Warning Protocol

- Hierarchically nesting elements (e.g. relation/block/tuple) can be locked with **intention locks** IS and IX
- Idea
  - start locking at the root (relation) level
  - to place an S or X lock on a subelement, first place a corresponding intention lock IS or IX the element itself
    - Warns others: "I'll be reading/writing some subelement of this element"



Example (T_1 reads t_2, T_2 writes R1)

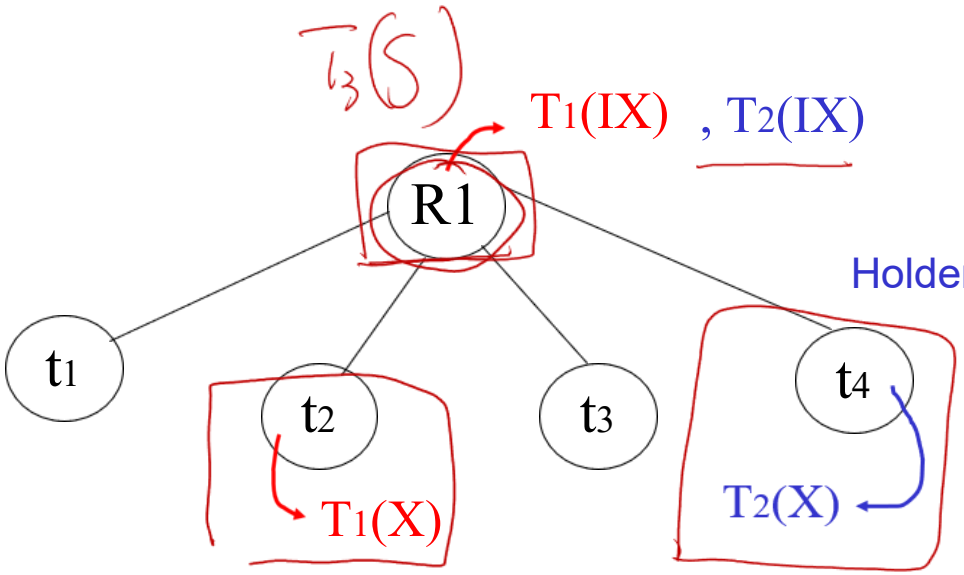


Comp Requestor

Holder	IS	IX	S	X
IS	T	T	T	F
IX				
S			T	F
X			F	F

As before

Example (T_1 writes t_2, T_2 writes to t_4)



Comp Requestor

Holder \ Requestor	IS	IX	S	X
IS	T	T	T	F
IX	T	T	F	F
S	T	F	T	F
X	F	F	F	F

As before

Compatibility of multiple granularity locks

Comp Requestor

Holder	IS	IX	S	X
IS	T	T	T	F
IX	T	T	F	F
S	T	F	T	F
X	F	F	F	F

SQL Server 完整的锁兼容性矩阵

	NL	SCH-S	SCH-M	S	U	X	IS	IU	IX	SIU	SIX	UIX	BU	RS-S	RS-U	RI-N	RI-S	RI-U	RI-X	RX-S	RX-U	RX-X
NL	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SCH-S	N	N	C	N	N	N	N	N	N	N	N	N	N	I	I	I	I	I	I	I	I	I
SCH-M	N	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
S	N	N	C	N	N	C	N	N	C	N	C	C	C	N	N	N	N	N	C	N	N	C
U	N	N	C	N	C	C	N	C	C	C	C	C	C	N	C	N	N	C	C	N	C	C
X	N	N	C	C	C	C	C	C	C	C	C	C	C	C	C	N	C	C	C	C	C	C
IS	N	N	C	N	N	C	N	N	N	N	N	N	C	I	I	I	I	I	I	I	I	I
IU	N	N	C	N	C	C	N	N	N	N	N	C	C	I	I	I	I	I	I	I	I	I
IX	N	N	C	C	C	C	N	N	N	C	C	C	C	I	I	I	I	I	I	I	I	I
SIU	N	N	C	N	C	C	N	N	C	N	C	C	C	I	I	I	I	I	I	I	I	I
SIX	N	N	C	C	C	C	N	N	C	C	C	C	C	I	I	I	I	I	I	I	I	I
UIX	N	N	C	C	C	C	N	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
BU	N	N	C	C	C	C	C	C	C	C	C	C	N	I	I	I	I	I	I	I	I	I
RS-S	N	I	I	N	N	C	I	I	I	I	I	I	I	N	N	C	C	C	C	C	C	C
RS-U	N	I	I	N	C	C	I	I	I	I	I	I	I	N	C	C	C	C	C	C	C	C
RI-N	N	I	I	N	N	N	I	I	I	I	I	I	I	C	C	N	N	N	N	C	C	C
RI-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	N	N	N	C	C	C	C
RI-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	N	N	C	C	C	C	C
RI-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	N	C	C	C	C	C	C
RX-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C

图例

N	不冲突	SIU	共享意向更新
I	非法	SIX	共享意向排他
C	冲突	UIX	更新意向排他
		BU	大容量更新
NL	没有锁	RS-S	共享范围-共享
SCH-S	架构稳定性锁	RS-U	共享范围-更新
SCH-M	架构修改锁	RI-N	插入范围-空
S	共享	RI-S	插入范围-共享
U	更新	RI-U	插入范围-更新
X	排他	RI-X	插入范围-排他
IS	意向共享	RX-S	排他范围-共享
IU	意向更新	RX-U	排他范围-更新
IX	意向排他	RX-X	排他范围-Exclusive

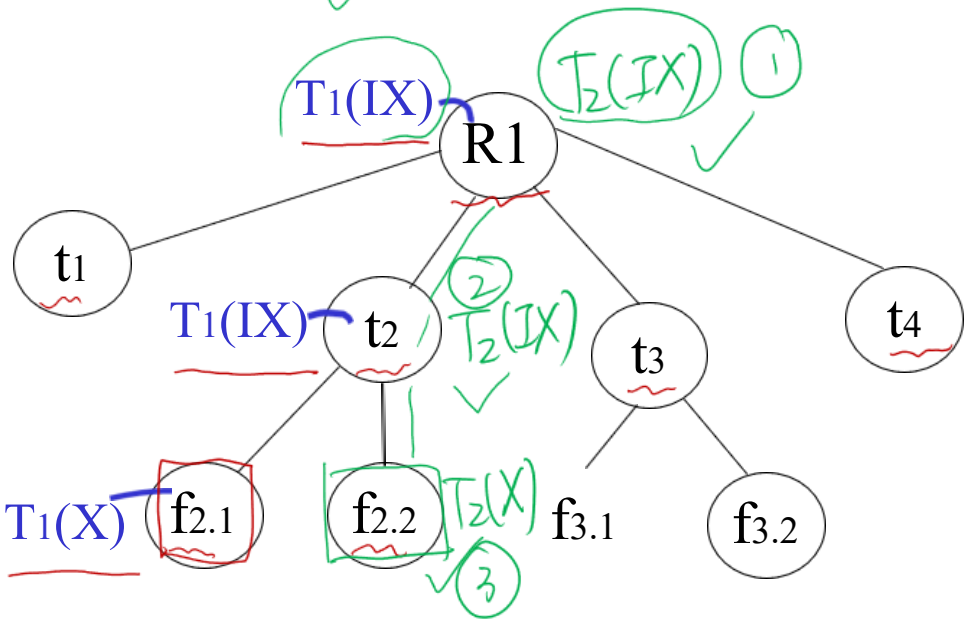
Database System - Nanka

Rules

(1) Follow multiple granularity comp function
(2) Lock root of tree first, any mode
(3) Node Q can be locked by Ti in S or IS only if parent(Q) can be locked by Ti in IX or IS
(4) Node Q can be locked by Ti in X,IX only if parent(Q) locked by Ti in IX
(5) Ti is two-phase
(6) Ti can unlock node Q only if none of Q'schildren are locked by Ti

Exercise:

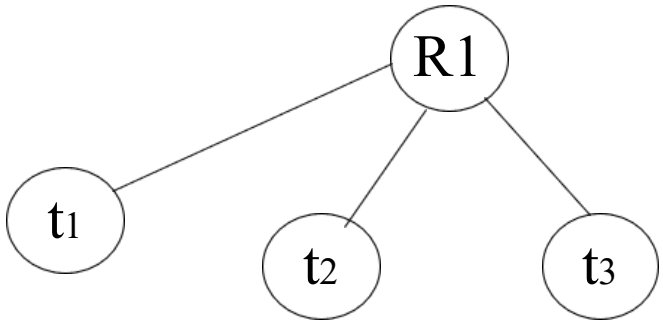
- Can T2 write element f2.2? What locks will T2 get?



Deletions and Phantom tuples

- Get an exclusive lock on A before deleting A
- Insertions more problematic:
  - possible to lock only existing elements
- **Phantom tuples**:
  - tuples that should have been locked, but did not exist when the locks were taken

- Use multiple granularity tree
- Before insert of node Q, lock parent(Q) in X mode



Summary-more lock

- 了解锁的多种类型，重点掌握共享锁和排它锁，可以在一个事务中添加这些锁的动作，使其满足规则1和规则3
- 掌握带有读锁和写锁的事务如何生成合法的可串行化调度
- 了解DBMS系统是如何实现两段锁协议的
- 掌握意向锁的基本原理

Database System - Nankai

多选题 1分

互动交流三 —不定项选择题

如果数据库元素的层次关系和持有锁的情况如图所示，

T_2事务要写$f_{2.2}$元素，能否成功?

A YES　　B NO

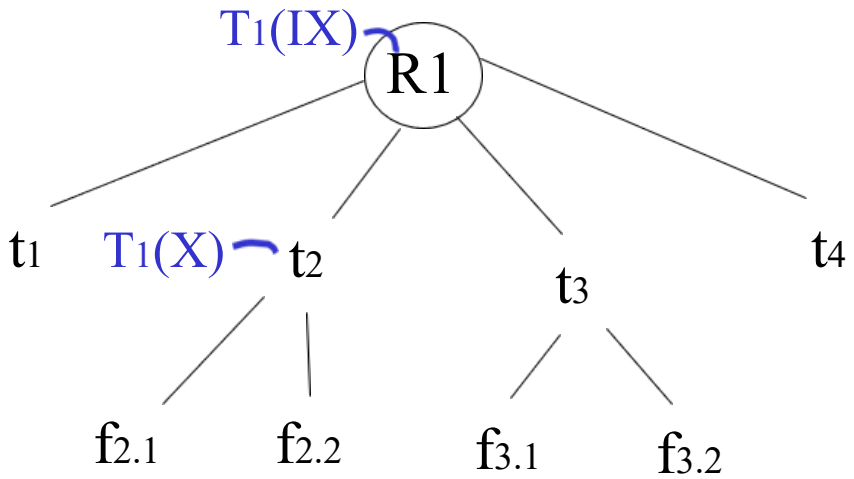
T_2事务要写$f_{2.2}$元素，应该申请哪些锁?

C R1的 IX锁

D t_2的 IX锁

E t_2的 X锁

F $F_{2.2}$ 的X锁



提交

tabase System - Nankai

多选题 1分

互动交流四 —不定项选择题

如果数据库元素的层次关系和持有锁的情况如图所示，

T_2事务要写$f_{3.2}$元素，能否成功?

A YES　　B NO

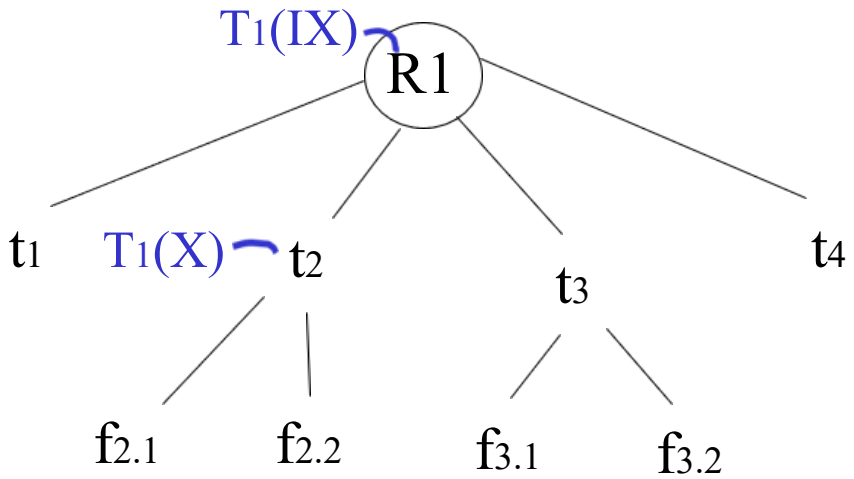
T_2事务要写$f_{3.2}$元素，应该申请哪些锁?

C R1的 IX 锁

D t_3的 IX 锁

E $F_{3.2}$ 的 IX 锁

F $F_{3.2}$ 的 X 锁



提交

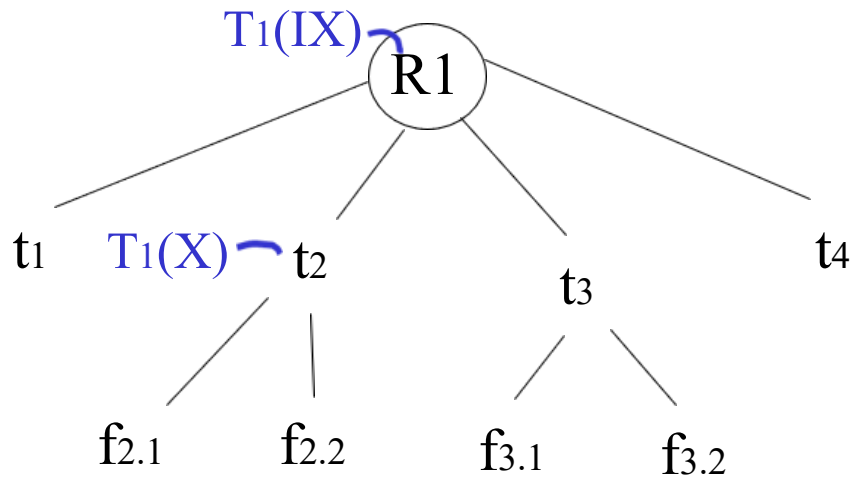
tabase System - Nankai

多选题 1分

互动交流五 —不定项选择题

如果数据库元素的层次关系和持有锁的情况
如右图所示，T_2事务插入T_5元组，能否成功?

A YES　　B NO



T_2事务应该对R1申请什么锁?

C R1的 IX 锁

D R1的 X 锁

提交

tabase System - Nankai

Week16_Course

Database System
Concurrency Control part5-1

Timestamp and Validation

Database System - Nankai

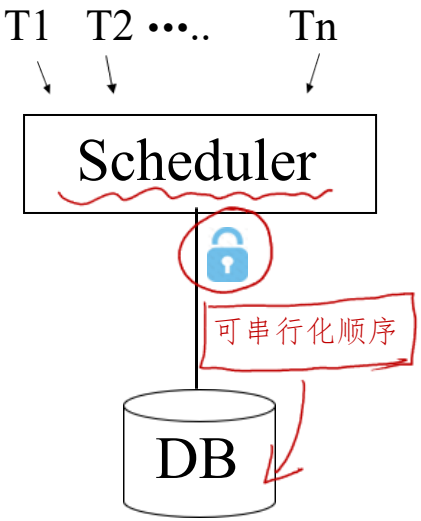
Concurrency Control

- Schedule
- Conflict Serializable Schedule
- Two phase locking
- Warning Protocol
- Timestamp and Validation

防止出现不可串行化的主要技术

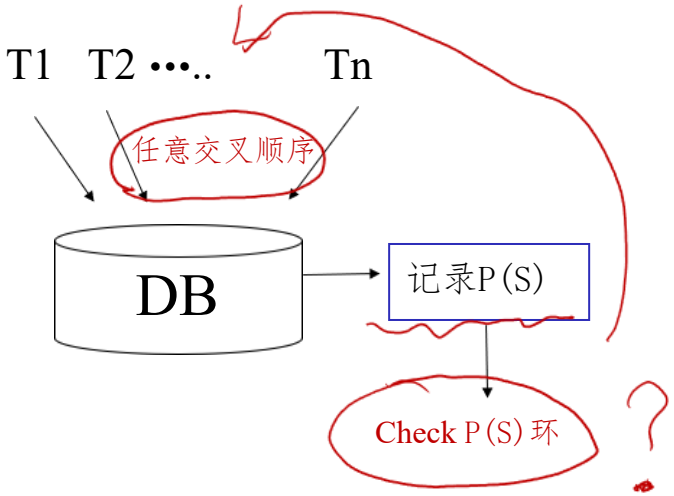
Pessimistic strategy: LOCK

Prevent occurrence of cycles in P(S)



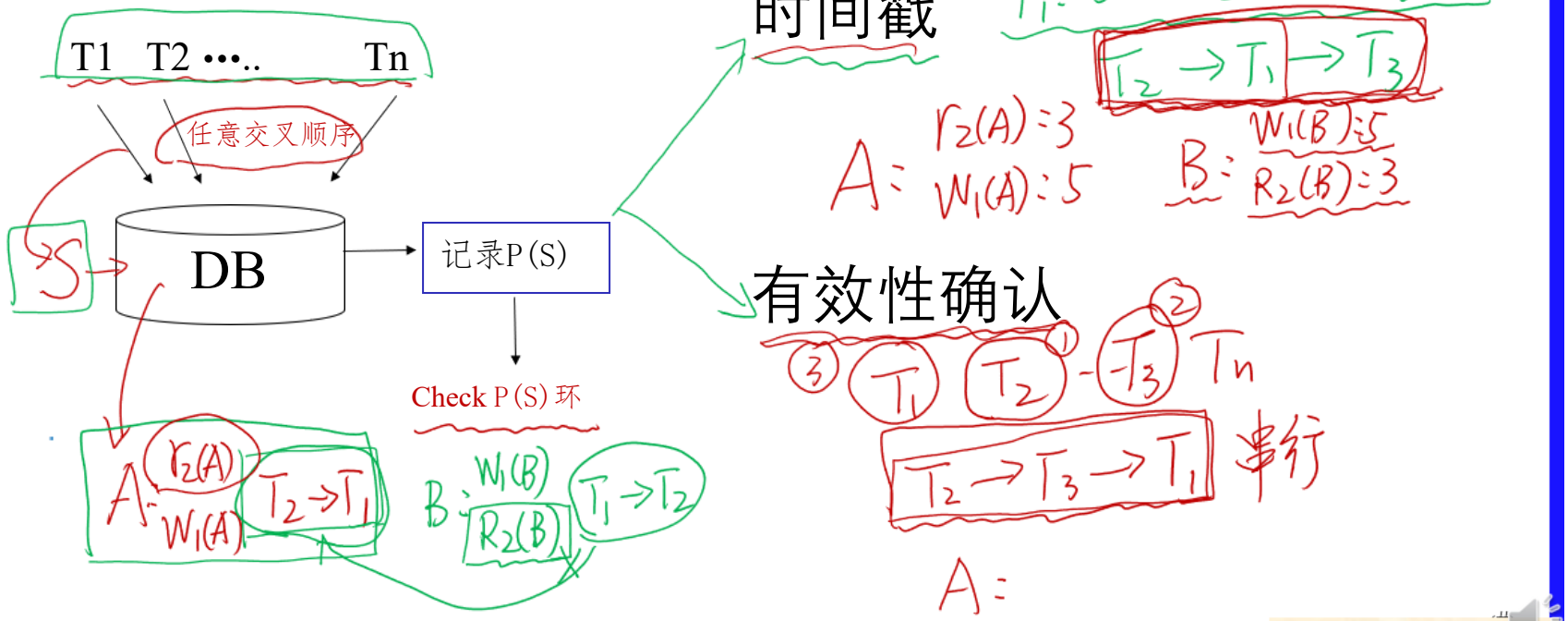
Optimistic strategy

Run system, recording P(S);
check P(S) for cycles,



Database System - Nankai

保证事务可串行化的其他方法



保证事务可串行化的其他方法

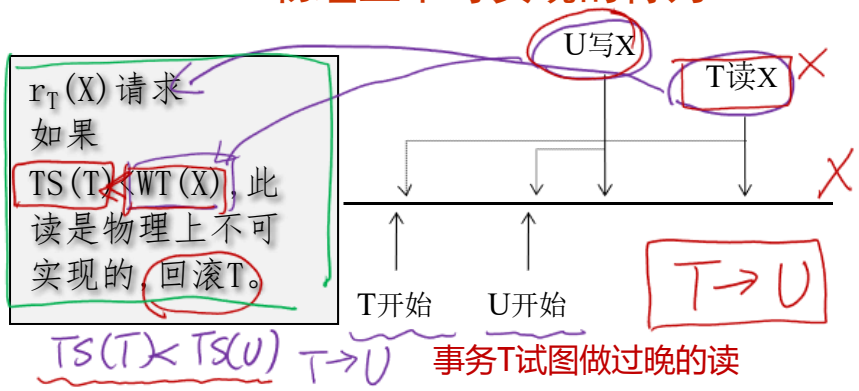
- 时间戳
  - 每个事务分配一个“时间戳”
  - 记录最后读和写每个数据库元素的事务时间戳
  - 比较上述值以保证按照事务的时间戳排序的串行调度
- 有效性确认
  - 当事务将要提交时，检查事务的时间戳和数据库元素
  - 按照有效性确认时间对事务进行排序的串行调度

中止并重启试图参与非可串行化行为的事务

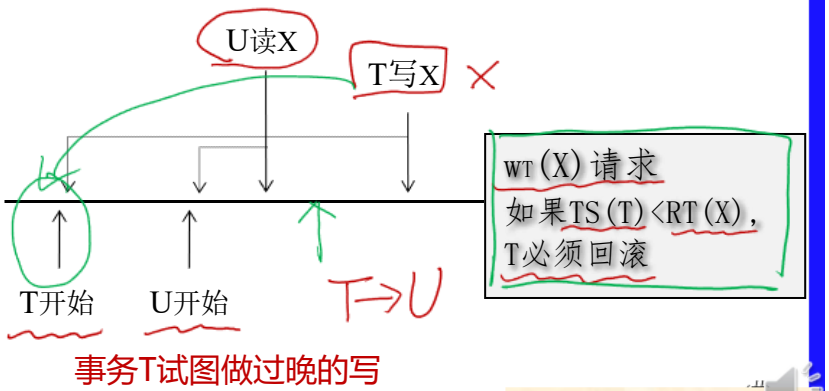
时间戳

- 调度器赋给每个事务T一个唯一的数值TS(T)
- 每个数据库元素X上有两个时间戳和一个附加位
  - RT(X) 读X的事务中最高的时间戳
  - WT(X) 写X的事务中最高的时间戳
  - C(X) X的提交位，该位为真当且仅当最近写X的事务已经提交

物理上不可实现的行为：



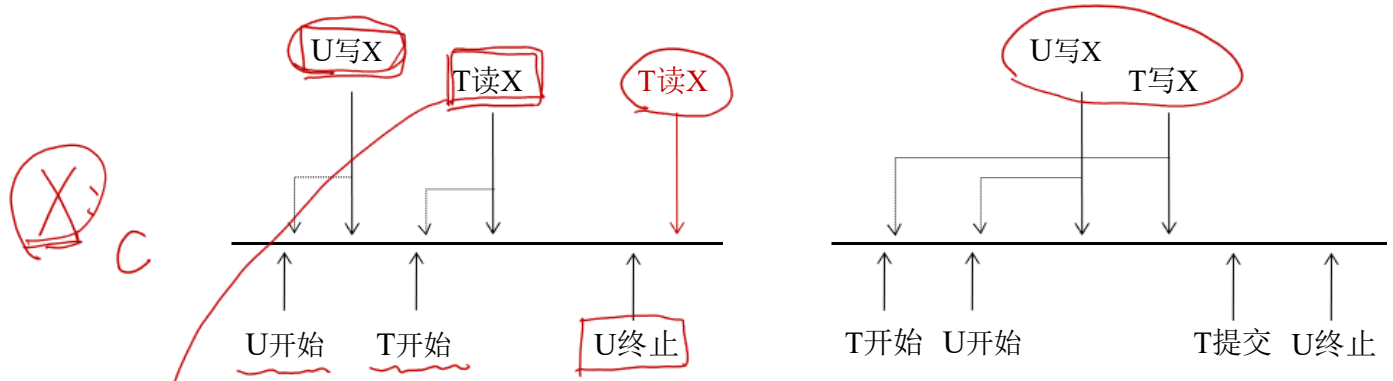
事务T试图做过晚的读



事务T试图做过晚的写

Database System - Nankai

脏数据问题



C(X)为假

$r_T(X)$请求

a. 如果TS(T)>=WT(X),此读是物理上可实现的

1. 如果C(X)为真，同意请求。如果TS(T)>RT(X)，置RT(X):=TS(T)
2. 如果C(X)为假，推迟T直到C(X)为真或写X的事务终止

Database System - Nankai

基于时间戳调度的规则

1. $r_T(X)$请求
   a. 如果TS(T)>=WT(X),此读是物理上可实现的
      1. 如果C(X)为真，同意请求。如果TS(T)>RT(X)，置RT(X):=TS(T)
      2. 如果C(X)为假，推迟T直到C(X)为真或写X的事务终止
   a. 如果TS(T)<WT(X),此读是物理上不可实现的。回滚T。
1. $w_T(X)$请求
   a. 如果TS(T)>=RT(X)并且TS(T)>=WT(X),物理上可实现,必须执行为X写入新值；置WT(X):=TS(T)并且置C(X):=false
   b. 如果TS(T)>=RT(X)但是TS(T)<WT(X),物理上可实现……
   c. 如果TS(T)<RT(X), T必须回滚
1. 提交T请求……C(X):=true
2. 终止T请求……

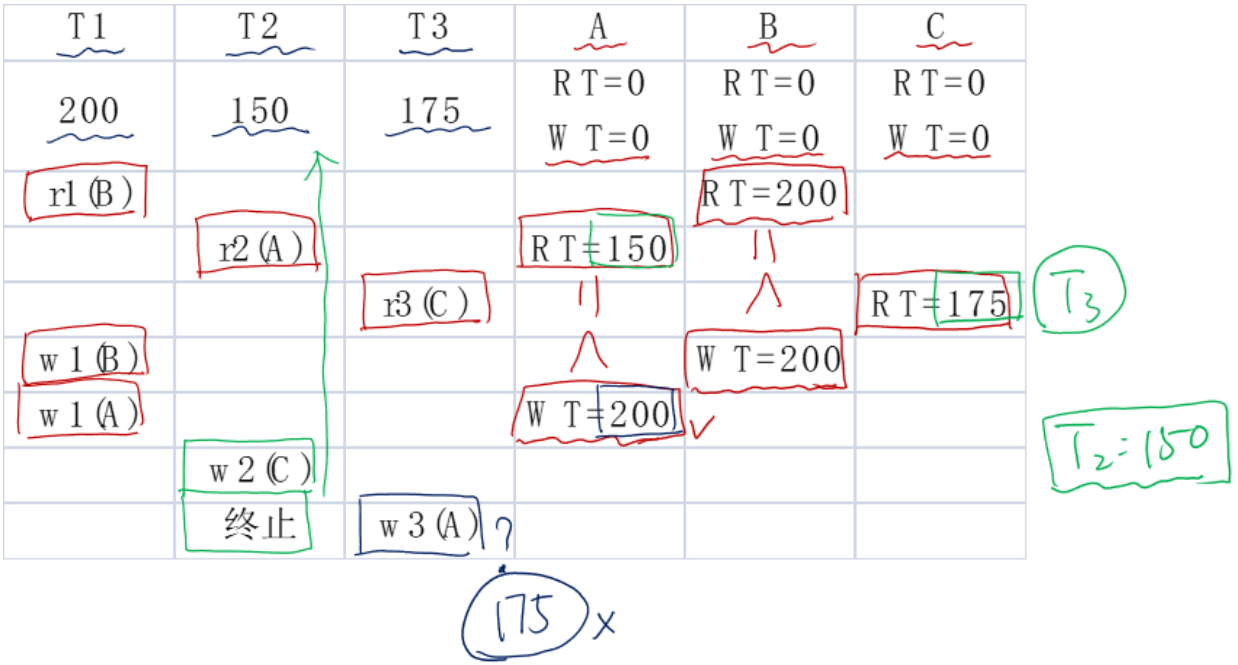
rT(X)请求
a. 如果TS(T)>=WT(X),此读是物理上可实现的
1. 如果C(X)为真，同意请求。如果TS(T)>RT(X)，置RT(X):=TS(T)
2. 如果C(X)为假，推迟T直到C(X)为真或写X的事务终止

$R_T(X)$请求
如果TS(T)<WT(X),此读是物理上不可实现的,回滚T。

wT(X)请求
如果TS(T)<RT(X),
T必须回滚

时间戳示例

T_2, T_3, T_1



多版本时间戳

T1	T2	T3	T4	A
150	200	175	225	RT=0 W T=0
r1(A)				RT=150
w 1(A)				W T=150
	r2(A)			RT=200
	w 2(A)			W T=200
		r3(A)		
		终止	r4(A)	RT=225

175

T1	T2	T3	T4	A_0	A_{150}	A_{200}
150	200	175	225			
r1(A)				读		
w 1(A)					创建	
	r2(A)				读	
	w 2(A)					创建
		r3(A)	175		读	
			r4(A) 225			读

时间戳与封锁

- 封锁在事务等待锁时会频繁地推迟事务
- 如果并发事务频繁读写公共元素，回滚会很频繁

商用系统中有一个折中方案，调度器将事务分为只读事务和读/写事务

✓ 只读事务使用多版本时间戳执行

✓ 读/写事务使用两阶段封锁执行

多选题 1分

互动交流一 一不定项选择题

以下哪些机制是DBMS采用的并发控制策略?

A 采用两阶段锁的机制，控制并发事务的执行

B 采用串行调度顺序，控制并发事务的执行

C 按照事务的时间戳顺序，形成串行调度执行

D 按照事务有效性确认顺序，形成串行调度执行

提交

se System - Nankai

多选题 1分

互动交流二

以下哪些策略，可能会造成数据库事务的频繁回滚?

A 采用两阶段锁的机制，控制并发事务的执行

B 采用串行调度顺序，控制并发事务的执行

C 按照事务的时间戳顺序，形成串行调度执行

D 按照事务有效性确认顺序，形成串行调度执行

提交

se System - Nankai

多选题 1分

互动交流三

以下哪种机制，会提高只读事务的并发度?

A 两阶段封锁

B 时间戳

C 多版本时间戳

D 任意机制都可以

提交

se System - Nankai