CSE 444: Database Internals

Section 7:
Concurrency

Today

- Timestamp-based Concurrency Control
- Multi-version Concurrency Control

Problem 1: Timestamp-based Concurrency Control

Timestamp-based Concurrency Control

- Some transaction, T.
- Some element (tuple/page), X.

- **TS**(T) - timestamp for transaction T
  - Stays constant for all of T's operations

- **WT**(X) – latest write timestamp for X
  - Set WT(X) = TS(T)

- **RT**(X) – latest read timestamp for X
  - Set RT(X) = TS(T)

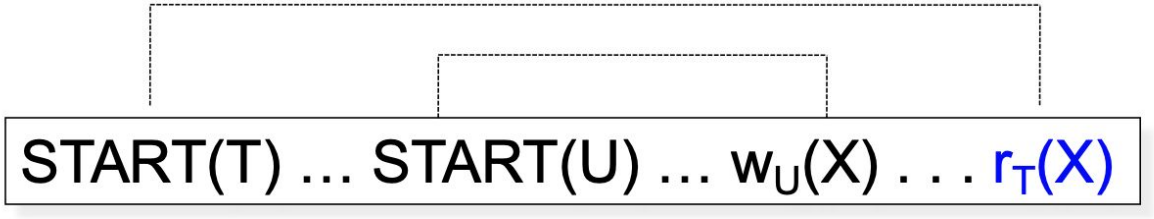
- **C**(X) – X's value has been committed
  - 1 if true, 0 if not

Timestamp-based Concurrency Control

- **Actions for transaction T**
  - **Grant** a read/write request for a transaction
  - **Abort** (in case T violates physical reality – late actions)
  - **Delay** (make the Grant or Abort decision later)
    - When writing, the change is always tentative until we decide to commit. For this, we use a commit bit C to keep track if the transaction that last wrote X has committed
  - **Ignore *Thomas Write Rule*** – ignore outdated writes

Timestamp-based Concurrency Control - Four Rules

- **Rule 1**: **Read** request on X by T

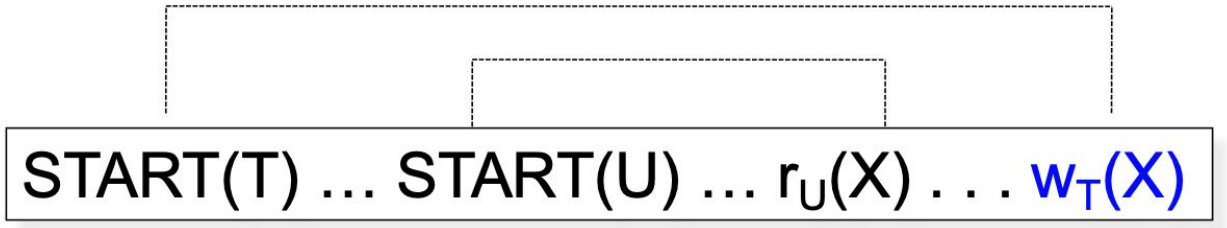


  - TS(T) < WT(X), **abort**, (read too late)

  - TS(T) >= WT(X), physically realizable
    - If C = 1, **grant**, update RT(X)
    - If C = 0, **delay** T

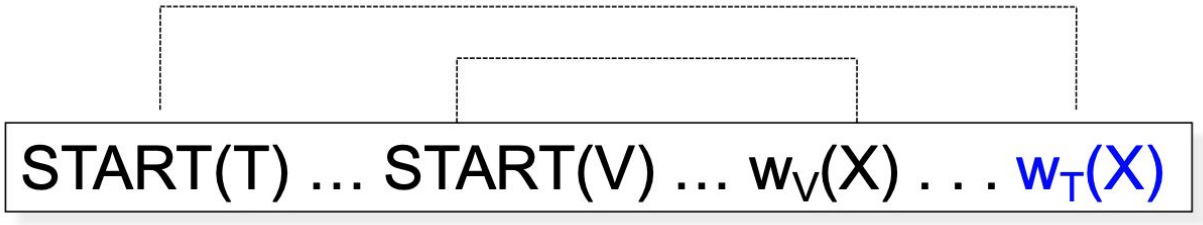
Timestamp-based Concurrency Control - Four Rules

- **Rule 2**: **Write** request on X by T



  - TS(T) < RT(X) (write too late)
    - **Abort**

- TS(T) >= RT(X), physically realizable
  - TS(T) >= WT(X)
    - then **grant**, update WT(X), set C = 0 (as it's not committed yet)



  - TS(T) < WT(X)
    - If C = 1, **don't write X at all!** (***Thomas Write Rule*** – ignore outdated writes)
    - If C = 0, **delay**

Timestamp-based Concurrency Control - Four Rules

- **Rule 3**: **Commit** request by T
  - Set C = 1 for all X written by T
  - Allow waiting transactions to proceed
- **Rule 4**: **Abort** transaction T
  - Check if the waiting transactions can proceed now.

Summary

Transaction wants to READ element X

If WT(X) > TS(T) then ROLLBACK
Else If C(X) = false, then WAIT
Else READ and update RT(X) to larger of TS(T) or RT(X)

Transaction wants to WRITE element X

If RT(X) > TS(T) then ROLLBACK
Else if WT(X) > TS(T)
Then If C(X) = false then WAIT
else IGNORE write (Thomas Write Rule)
Otherwise, WRITE, and update WT(X)=TS(T), C(X)=false

Timestamp-based Concurrency Control

Two transactions get started.

- $Start(T_1)$ -> $Start(T_2)$

Timestamp-based Concurrency Control

What will happen at the last request?

- $S_{tart}(T_1)$ -> $S_{tart}(T_2)$ -> $R_{T1}(A)$ -> $R_{T2}(A)$ -> $W_{T1}(B)$ -> **$W_{T2}(B)$**

Timestamp-based Concurrency Control

What will happen at the last request?

- $Start(T_1)$ -> $Start(T_2)$ -> $R_{T1}(A)$ -> $R_{T2}(A)$ -> $W_{T1}(B)$ -> **$W_{T2}(B)$**
  - **ACCEPTED**

Timestamp-based Concurrency Control

What will happen at the last request?

- $Start(T_1)$ -> $Start(T_2)$ -> $R_{T1}(A)$ -> $R_{T2}(A)$ -> $W_{T1}(B)$ -> **$W_{T2}(B)$**
  - **ACCEPTED**

- $Start(T_1)$ -> $Start(T_2)$ -> $R_{T2}(A)$ -> $Commit_{T2}$ -> $R_{T1}(A)$ -> **$W_{T1}(A)$**

Timestamp-based Concurrency Control

What will happen at the last request?

- $Start(T_1)$ -> $Start(T_2)$ -> $R_{T1}(A)$ -> $R_{T2}(A)$ -> $W_{T1}(B)$ -> **$W_{T2}(B)$**
  - **ACCEPTED**

- $Start(T_1)$ -> $Start(T_2)$ -> $R_{T2}(A)$ -> $Commit_{T2}$ -> $R_{T1}(A)$ -> **$W_{T1}(A)$**
  - **ABORT** T_1 because $R_{T2}(A)$ precedes

Problem 1: Timestamp-based Concurrency Control

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$						

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		

1. Physically realizable:
$TS(T_1) >= WT(X)$

2. C = 1: grant request

3. Update RT : $TS(T_1) > RT(X)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$					

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$					

1. Physically realizable:
$TS(T_2) >= RT(X)$

2. Set C = 0 (uncommitted)

3. Update WT

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$						

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						

1. **NOT** Physically realizable:
$TS(T_1) < RT(X)$

Abort/rollback

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$				

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	

1. Physically realizable:
$TS(T_3)$ >= RT(Y) and $TS(T_3)$ >= WT(Y)

2. Update WT and C (not committed yet)

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$					

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					

1. Physically realizable:
$TS(T_2)$ >= RT(Y) although $TS(T_2) < WT(Y)$
2. We could not apply Thomas' write rule **(ignore $W_2(Y)$)** since C=0

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					
		C_3				

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					
		C_3			C=1	

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					
		C_3			C=1	

A later write by T_3 has been committed!

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					
		C_3			C=1	
	Ignore $W_2(Y)$ and proceed					

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
	Ignore $W_2(Y)$ and **proceed**					
			$W_4(Z)$			

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
	Ignore $W_2(Y)$ and **proceed**					
			$W_4(Z)$			WT=4, C = 0

1. Physically realizable:
$TS(T_4)$ >= RT(Z) and $TS(T_4)$ >= WT(Z)

2. Update WT and C (not committed yet)

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
	Ignore $W_2(Y)$ and **proceed**					
			$W_4(Z)$			WT=4, C = 0
			C_4			C=1

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
	Ignore $W_2(Y)$ and **proceed**					
			$W_4(Z)$			WT=4, C = 0
			C_4			C=1
	$R_2(Z)$					

1. **NOT** Physically realizable: $TS(T_2) < WT(Z)$

Abort/rollback

			T4	X	Y	Z
			4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
	d **proceed**					
			$W_4(Z)$			WT=4, C = 0
			C_4			C=1
	$R_2(Z)$: **abort**					

Timestamp-based Concurrency Control

Questions?

Multiversion Concurrency Control

- Maintains **old** versions of database elements in addition to the current version in the database itself.

- The idea is to allow reads that would otherwise result in an abort (as the current version was written by future transaction)

Problem with Timestamp-Based Scheduling

T1	T2	T3	T4	A
150	200	175	225	RT = 0 WT = 0
$R_1(A)$				RT = 150
$W_1(A)$				WT = 150
	$R_2(A)$			RT = 200
	$W_2(A)$			WT = 200
		$R_3(A)$		
		Abort		
			$R_4(A)$	RT = 225

Had to abort because WT(A) is greater than my own timestamp

Would have been useful if I had access to an old version of A (from 150)…

Multiversion Timestamps

T1	T2	T3	T4	A_0	A_{150}	A_{200}
150	200	175	225	RT = 0 WT = 0		
$R_1(A)$				RT = 150		
$W_1(A)$					Create	
	$R_2(A)$				RT=200	
	$W_2(A)$					Create
		$R_3(A)$			RT=200 RT= max(175, 200)	
			$R_4(A)$			RT=225

Don't have to abort

Just read a previous value of A

Undo Logging

- Two Rules:
  - 1. If a transaction writes element **X**, then the log record of this update **<T,X,v>** must be written to disk before the new value of **X** is written to disk.
  - 2. If a transaction commits, then the **COMMIT** must be written to disk only after all elements changed by the transaction have been written to disk.

UNDO LOG RULES

1. <T, X, v> before OUTPUT(X)
2. OUTPUT(X) before <COMMIT>

Action				Disk A	Disk B	Log
						<START T>
INPUT(A				8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

When recovering (with UNDO logging)…

- We can not simply ignore the log before a recent commit
  - Many transactions interleave at once. If we truncate before a commit for a transaction, any information about those unfinished transactions would be lost.

- Instead, we can use checkpoint the log periodically…

Review: Checkpointing

- **Checkpointing (naïve)**
  - Write a <START CKPT(T1,…,Tk)>. Flush log to disk
  - **Stop accepting new transactions**
  - Wait until all active transactions abort/commit
  - Write <CKPT>. Flush log to disk.
  - Resume accepting transactions

- **Nonquiescent Checkpointing**
  - Write a <START CKPT(T1,…,Tk)>. Flush log to disk
  - Continue normal operation
  - When all of T1,…,Tk have completed, write <END CKPT>. Flush log to disk
  - **More efficient, system does not seem to be stalled**

Problem 1. UNDO Logging

```
LSN1      <START T1>
LSN2      <T1 X 5>
LSN3      <START T2>
LSN4      <T1 Y 7>
LSN5      <T2 X 9>
LSN6      <START T3>
LSN7      <T3 Z 11>
LSN8      <COMMIT T1>
LSN9      <START CKPT(T2,T3)>
LSN10 <T2 X 13>
LSN11 <T3 Y 15>
      *CRASH*
```

1.
Show how far back in the recovery manager needs to read the log

(which LSN do we need to read up to?)

UNDO: How far to scan log from the end?

- **Case 1**: See **<END CKPT>** first
  - All incomplete transactions began after <START CKPT…>
- **Case 2:** See **<START CKPT(T1..TK)>** first
  - Incomplete transactions began after <START CKPT…> or incomplete ones among T1.. TK
  - Find the earliest <START Ti> among them
  - At most we have to go until the previous <START CKPT>…<END CKPT>

```
LSN1      <START T1>
LSN2      <T1 X 5>
LSN3      <START T2>
LSN4      <T1 Y 7>
LSN5      <T2 X 9>
LSN6      <START T3>
LSN7      <T3 Z 11>
LSN8      <COMMIT T1>
LSN9      <START CKPT(T2,T3)>
LSN10     <T2 X 13>
LSN11     <T3 Y 15>
        *CRASH*
```

Problem 1. UNDO Logging

```
LSN1      <START T1>
LSN2      <T1 X 5>
LSN3      <START T2>
LSN4      <T1 Y 7>
LSN5      <T2 X 9>
LSN6      <START T3>
LSN7      <T3 Z 11>
LSN8      <COMMIT T1>
LSN9      <START CKPT(T2,T3)>
LSN10 <T2 X 13>
LSN11 <T3 Y 15>
      *CRASH*
```

1.
Show how far back in the recovery manager needs to read the log

(write the earliest LSN)

LSN3
(start of the earliest transaction among incomplete transactions)

Problem 1. UNDO Logging

```
LSN1      <START T1>
LSN2      <T1 X 5>
LSN3      <START T2>
LSN4      <T1 Y 7>
LSN5      <T2 X 9>
LSN6      <START T3>
LSN7      <T3 Z 11>
LSN8      <COMMIT T1>
LSN9      <START CKPT(T2,T3)>
LSN10 <T2 X 13>
LSN11 <T3 Y 15>
      *CRASH*
```

2.
Show the actions of the recovery manager during recovery.

Problem 1. UNDO Logging

```
LSN1      <START T1>
LSN2      <T1 X 5>
LSN3      <START T2>
LSN4      <T1 Y 7>
LSN5      <T2 X 9>
LSN6      <START T3>
LSN7      <T3 Z 11>
LSN8      <COMMIT T1>
LSN9      <START CKPT(T2,T3)>
LSN10 <T2 X 13>
LSN11 <T3 Y 15>
      *CRASH*
```

2.
Show the actions of the recovery manager during recovery.

Y = 15
X = 13
Z = 11
X = 9

T1	T2	T3	T4	A_0	A_{150}	A175	A_{200}
150	200	175	225	RT = 0 WT = 0			
$R_1(A)$				RT = 150			
$W_1(A)$					Create		
	$W_2(A)$						Create
		$R_3(A)$			RT=175		
		W3(A)				Create	
			$R_4(A)$				RT=225