

CSE 562 Database Systems

More About Transaction Processing

Some slides are based or modified from originals by
Database Systems: The Complete Book,
Pearson, Prentice Hall 2nd Edition,
©2006 Garcia-Molina, Ullman, and Widom

cse@buffalo

More on Transaction Processing

- Topics:
- Cascading rollback, recoverable schedule
 - Deadlocks
 - Prevention
 - Detection
 - View serializability
 - Distributed transactions
 - Long transactions (nested, compensation)

Concurrency Control & Recovery

Example:

T_j	T_i
\vdots	\vdots
$W_j(A)$	\vdots
\vdots	$ri(A)$
\vdots	Commit T_i
Abort T_j	\vdots

☛ Non-Persistent Commit (Bad!)

avoided by
recoverable
schedules

Concurrency Control & Recovery

Example:

T_j	T_i
\vdots	\vdots
$W_j(A)$	\vdots
\vdots	$ri(A)$
\vdots	Commit T_i
Abort T_j	\vdots

☛ Non-Persistent Commit (Bad!)

avoided by
recoverable
schedules

To model this, two new actions:

- C_i - transaction T_i commits
- A_i - transaction T_i aborts

UB CSE 562 Spring 2009

9

Back To Example:

$$\begin{array}{c} T_j \quad T_i \\ \hline \vdots \quad \vdots \\ W_j(A) \quad \vdots \\ \vdots \quad r_i(A) \\ \vdots \quad \vdots \\ C_i \leftarrow \text{can we commit here?} \end{array}$$

UB CSE 562 Spring 2009

10

Definition

T_i reads from T_j in S ($T_j \Rightarrow_S T_i$) if

- (1) $w_j(A) <_S r_i(A)$
- (2) $a_j \not<_S r_i(A)$ ($\not<_S$: does not precede)
- (3) If $w_j(A) <_S w_k(A) <_S r_i(A)$ then $a_k <_S r_i(A)$

UB CSE 562 Spring 2009

11

Definition

Schedule S is recoverable if
whenever $T_j \Rightarrow_S T_i$ and $j \neq i$ and $C_i \in S$
then $C_j <_S C_i$

UB CSE 562 Spring 2009

12

Note: in transactions, reads and writes precede commit or abort

⇔ If $C_i \in T_i$, then $ri(A) < C_i$

$wi(A) < C_i$

⇔ If $A_i \in T_i$, then $ri(A) < A_i$

$wi(A) < A_i$

- Also, one of C_i, A_i per transaction

How to achieve recoverable schedules?

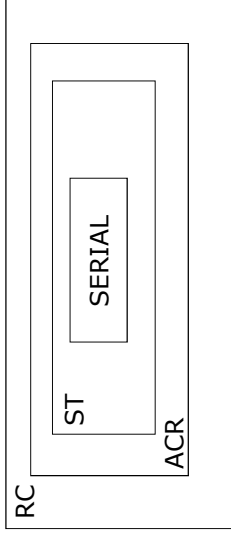
⇔ With 2PL, hold write locks to commit (strict 2PL)

T_j	T_i
⋮	⋮
$w_j(A)$	⋮
⋮	⋮
C_j	⋮
$u_j(A)$	⋮
⋮	$ri(A)$

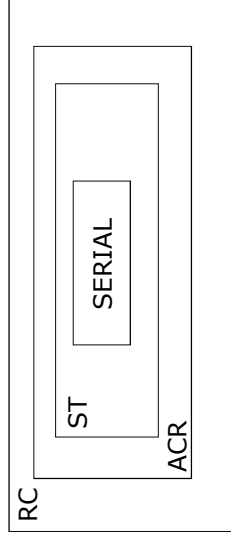
⇔ With validation, no change!

- S is recoverable if each transaction *commits* only after all transactions from which it read have committed.
- S avoids cascading rollback if each transaction may *read* only those values written by committed transactions.

- S is strict if each transaction may *read and write* only items previously written by committed transactions.



Where are serializable schedules?



Examples

- Recoverable:
 - $w_1(A) w_1(B) w_2(A) r_2(B) c_1 c_2$
- Avoids Cascading Rollback:
 - $w_1(A) w_1(B) w_2(A) c_1 r_2(B) c_2$
- Strict:
 - $w_1(A) w_1(B) c_1 w_2(A) r_2(B) c_2$

Assumes $w_2(A)$ is done without reading

Deadlocks

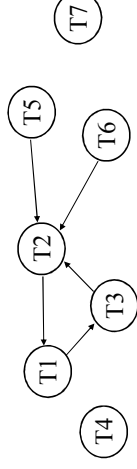
- Detection
 - Wait-For Graph
- Prevention
 - Resource Ordering
 - Timeout
 - Wait-Die
 - Wound-Wait

UB CSE 562 Spring 2009

21

Deadlock Detection

- Build Wait-For Graph
- Use lock table structures
- Build incrementally or periodically
- When cycle found, rollback victim



UB CSE 562 Spring 2009

22

Resource Ordering

- Order all elements A_1, A_2, \dots, A_n
- A transaction T can lock A_i after A_j only if $i > j$

Problem: Ordered lock requests not realistic in most cases

UB CSE 562 Spring 2009

23

Timeout

- If transaction waits more than L sec., roll it back!
- Simple scheme
- Hard to select L

UB CSE 562 Spring 2009

24

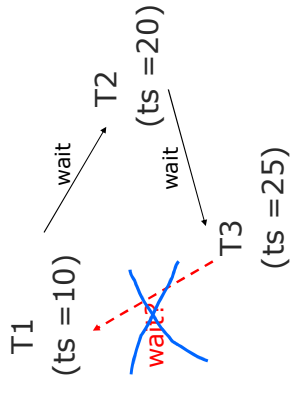
Wait-Die

- Transactions given a timestamp when they arrive ... $ts(T_i)$
- T_i can only wait for T_j if $ts(T_i) < ts(T_j)$...else die

UB CSE 562 Spring 2009

25

Example



UB CSE 562 Spring 2009

26

Starvation with Wait-Die

- When transaction dies, re-try later with what timestamp
 - original timestamp
 - new timestamp (time of re-submit)

UB CSE 562 Spring 2009

27

Starvation with Wait-Die

- Resubmit with original timestamp
- Guarantees no starvation
 - Transaction with oldest ts never dies
 - A transaction that dies will eventually have oldest ts and will complete...

UB CSE 562 Spring 2009

28

Second Example

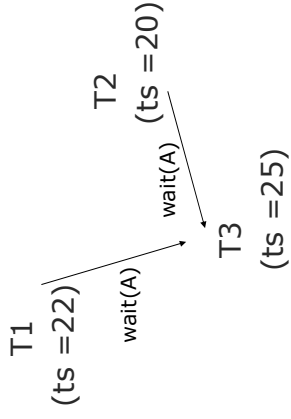


UB CSE 562 Spring 2009

29

Second Example (cont'd)

One option:
T1 waits just for T3, transaction holding lock.
But when T2 gets lock, T1 will have to die!

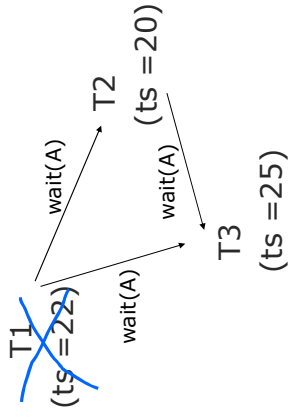


UB CSE 562 Spring 2009

30

Second Example (cont'd)

Another option:
T1 only gets A lock after T2, T3 complete,
so T1 waits for both T2, T3 \Rightarrow T1 dies right away!

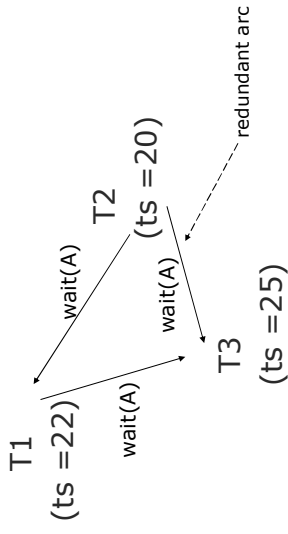


UB CSE 562 Spring 2009

31

Second Example (cont'd)

Yet another option:
T1 preempts T2, so T1 only waits for T3;
T2 then waits for T3 and T1... \Rightarrow T2 may starve?



UB CSE 562 Spring 2009

32

Wound-Wait

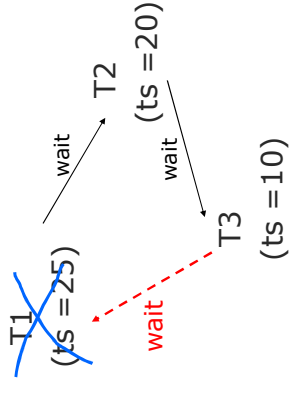
- Transactions given a timestamp when they arrive ... $ts(T_i)$
- T_i wounds T_j if $ts(T_i) < ts(T_j)$
else T_i waits

“Wound”: T_j rolls back and gives lock to T_i

UB CSE 562 Spring 2009

33

Example



UB CSE 562 Spring 2009

34

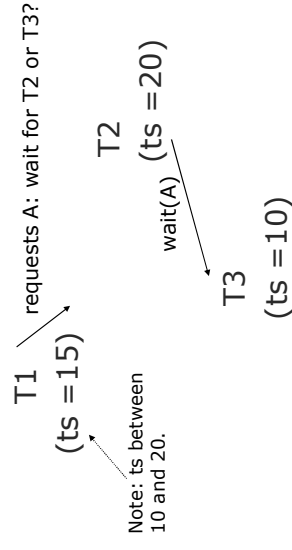
Starvation with Wound-Wait

- When transaction dies, re-try later with what timestamp?
 - original timestamp
 - new timestamp (time of re-submit)

UB CSE 562 Spring 2009

35

Second Example

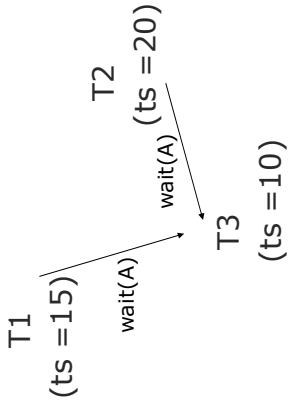


UB CSE 562 Spring 2009

36

Second Example (cont'd)

One option:
T1 waits just for T3, transaction holding lock.
But when T2 gets lock, T1 waits for T2 and wounds T2.

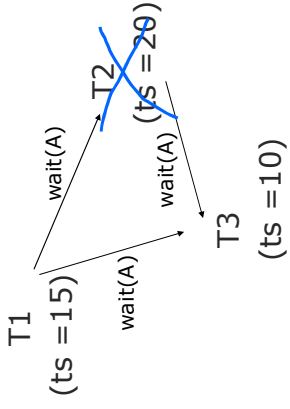


UB CSE 562 Spring 2009

37

Second Example (cont'd)

Another option:
T1 only gets A lock after T2, T3 complete,
so T1 waits for both T2, T3 \Rightarrow T2 wounded right away!

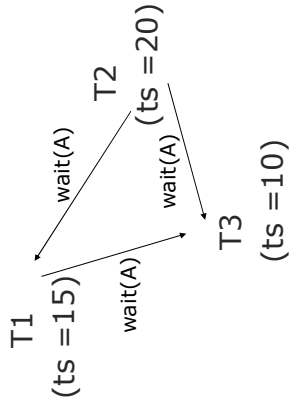


UB CSE 562 Spring 2009

38

Second Example (cont'd)

Yet another option:
T1 preempts T2, so T1 only waits for T3;
T2 then waits for T3 and T1... \Rightarrow T2 is spared!



UB CSE 562 Spring 2009

39

User/Program Commands

- Lots of variations, but in general
- Begin_work
 - Commit_work
 - Abort_work

UB CSE 562 Spring 2009

40

Nested Transactions

```
User program:  
  ⋮  
  Begin_work;  
  ⋮  
  ⋮  
  If results_ok, then commit work  
  else abort_work
```

UB CSE 562 Spring 2009

41

Nested Transactions

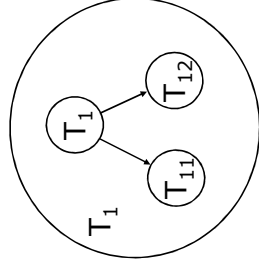
```
User program:  
  ⋮  
  Begin_work;  
  Begin_work;  
  ⋮  
  ⋮  
  If results_ok, then commit work  
  else {abort_work; try something else...}  
  ⋮  
  If results_ok, then commit work  
  else abort_work
```

UB CSE 562 Spring 2009

42

Parallel Nested Transactions

```
T1: begin-work  
  parallel:  
    T11: begin_work  
        commit_work  
    T12: begin_work  
        commit_work  
  commit_work
```



UB CSE 562 Spring 2009

43

Locking

Locking

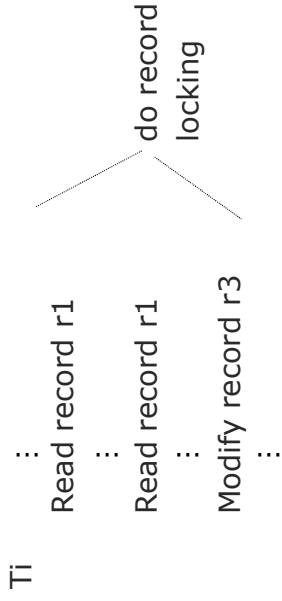
What are we really locking?



UB CSE 562 Spring 2009

44

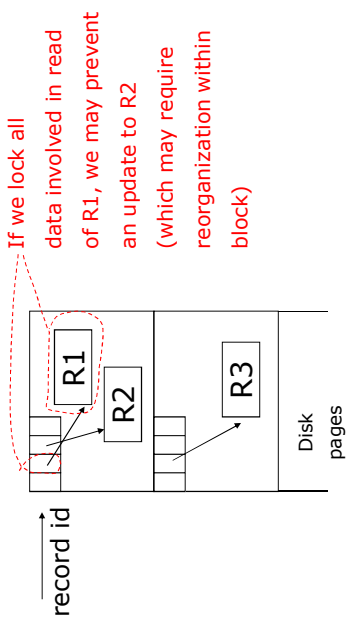
Example



UB CSE 562 Spring 2009

45

But Underneath



UB CSE 562 Spring 2009

46

Solution: View DB At Two Levels

Top level: record actions
record locks
undo/redo actions — logical

e.g., Insert record(X,Y,Z)
Redo: insert(X,Y,Z)
Undo: delete

UB CSE 562 Spring 2009

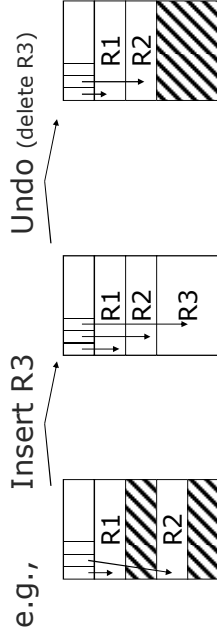
47

Low level: deal with physical details
latch page during action
(release at end of action)

UB CSE 562 Spring 2009

48

Note: undo does not return physical DB to original state; only same logical state



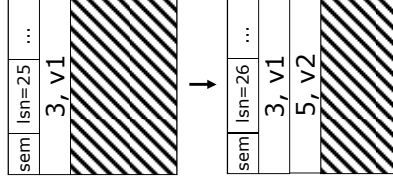
Logging Logical Actions

- Logical action typically span one block (physiological actions)
- Undo/redo log entry specifies undo/redo logical action
- Challenge: making actions idempotent
 - Example (bad): redo insert \Rightarrow key inserted multiple times!

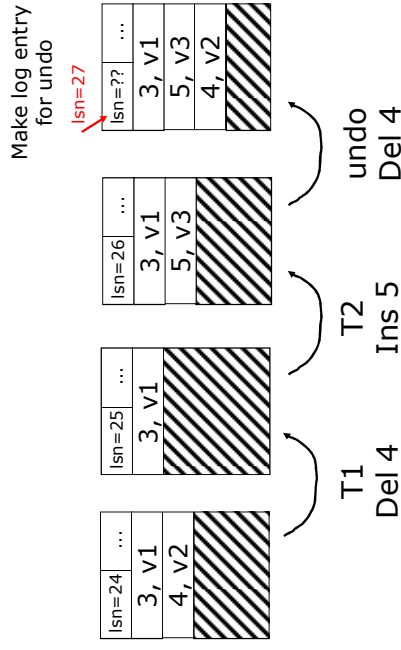
Solution: Add Log Sequence Number

Log record:

- LSN=26
- OP=insert(5,v2) into P
- ...



Still Have a Problem!



Compensation Log Records

- Log record to indicate undo (not redo) action performed
- Note: Compensation may not return page to exactly the initial state

UB CSE 562 Spring 2009

53

At Recovery: Example

Log:

lsn=21 T1 a1 p1	lsn=27 T1 a2 p2	lsn=35 T1 a2 ⁻¹ p2
...

UB CSE 562 Spring 2009

54

What To Do With p2 (During T1 Rollback)?

- If $lsn(p2) < 27$ then ... ?
- If $27 \leq lsn(p2) < 35$ then ... ?
- If $lsn(p2) \geq 35$ then ... ?

Note: $lsn(p2)$ is lsn of p copy on disk

UB CSE 562 Spring 2009

55

Recovery Strategy

- [1]** Reconstruct state at time of crash
- Find latest valid checkpoint, C_k , and let ac be its set of active transactions
 - Scan log from C_k to end:
 - For each log entry $[lsn, page]$ do:
 - if $lsn(page) < lsn$ then redo action
 - If log entry is start or commit, update ac

UB CSE 562 Spring 2009

56

Recovery Strategy

[2] Abort uncommitted transactions

- Set ac contains transactions to abort
- Scan log from end to C_k :
 - For each log entry (not undo) of an ac transaction, undo action (making log entry)
- For ac transactions not fully aborted, read their log entries older than C_k and undo their actions

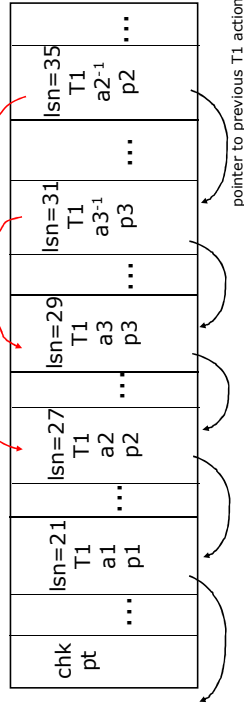
Example: What To Do After Crash

Log:

chk	lsn=21	lsn=27	lsn=29	lsn=31	lsn=35
pt	T1	T1	T1	T1	T1
...	a1	a2	a3	a3 ⁻¹	a2 ⁻¹
...	p1	p2	p3	p3	p2
...

During Undo: Skip Undo's

Log:



Related idea: Sagas

- Long running activity: T_1, T_2, \dots, T_n
- Each step/transaction T_i has a compensating transaction T_{i-1}
- Semantic atomicity: execute one of
 - T_1, T_2, \dots, T_n
 - $T_1, T_2, \dots, T_{n-1}, T_{n-2}, \dots, T_{n-1}$
 - $T_1, T_2, \dots, T_{n-2}, T_{n-1}, T_{n-3}, \dots, T_{n-1}$
 - ...
 - T_1, T_{n-1}
 - nothing

Summary

- Cascading rollback
- Recoverable schedule
- Deadlock
 - Prevention
 - Detection
- Nested transactions
- Multi-level view