

CS 245: Database System Principles

Notes 10: More TP

Hector Garcia-Molina

CS 245

Notes 10

1

Sections to Skim:

- Section 18.8 [18.8]
- Sections 19.2 19.4, 19.5, 19.6 [none, i.e., read all Ch 19]
- [In the Second Edition, skip all of Chapter 20, and Sections 21.5, 21.6, 21.7, 22.2 through 22.7]

CS 245

Notes 10

2

Chapter 19 [19] More on transaction processing

Topics:

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

CS 245

Notes 10

3

Concurrency control & recovery

Example:

T_j	T_i
⋮	⋮
$W_j(A)$	⋮
⋮	$r_i(A)$
⋮	Commit T_i
⋮	⋮
Abort T_j	⋮

CS 245

Notes 10

4

Concurrency control & recovery

Example:

T_j	T_i
⋮	⋮
$W_j(A)$	⋮
⋮	$r_i(A)$
⋮	Commit T_i
⋮	⋮
Abort T_j	⋮

☛ Non-Persistent Commit (Bad!)

CS 245

Notes 10

5

Concurrency control & recovery

Example:

T_j	T_i
⋮	⋮
$W_j(A)$	⋮
⋮	$r_i(A)$
⋮	Commit T_i
⋮	⋮
Abort T_j	⋮

☛ Non-Persistent Commit (Bad!)

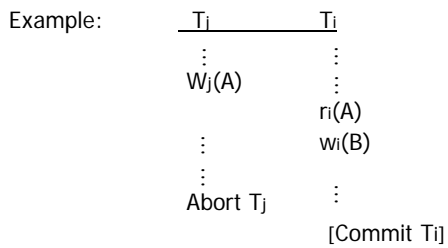
avoided by recoverable schedules

CS 245

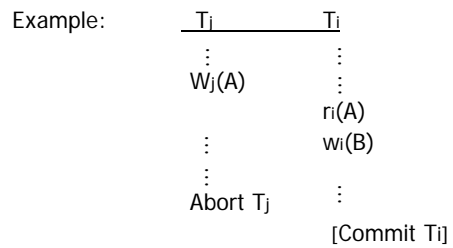
Notes 10

6

Concurrency control & recovery

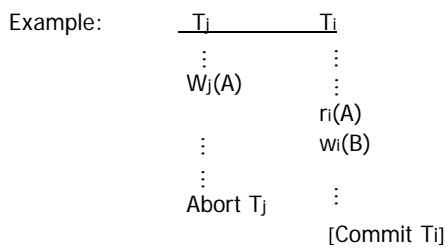


Concurrency control & recovery



➡ Cascading rollback (Bad!)

Concurrency control & recovery



➡ Cascading rollback (Bad!)

avoided by
[avoids-cascading-rollback \(ACR\)](#)
 schedules

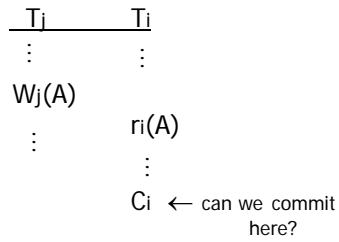
- Schedule is conflict serializable
- T_j → T_i
- But not recoverable

- Need to make "final" decision for each transaction:
 - **commit decision** - system guarantees transaction will or has completed, no matter what
 - **abort decision** - system guarantees transaction will or has been rolled back (has no effect)

To model this, two new actions:

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

Back to example:



Definition

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

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

Definition

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

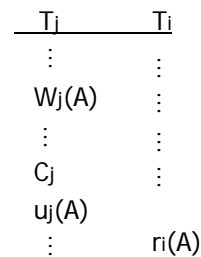
Note: in transactions, reads and writes precede commit or abort

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

- Also, one of C_i, A_i per transaction

How to achieve recoverable schedules?

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



★ With validation, no change!

- S is recoverable if each transaction *commits* only after all transactions from which it read have committed.

- 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 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.

- Relationship of RC, ACR, Strict



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 $w_1(A)$ is done without reading

Where are serializable schedules?



CS 245

Notes 10

25

Deadlocks

- Detection
 - Wait-for graph
- Prevention
 - Resource ordering
 - Timeout
 - Wait-die
 - Wound-wait

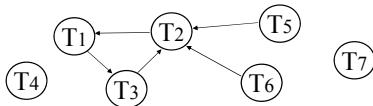
CS 245

Notes 10

26

Deadlock Detection

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



CS 245

Notes 10

27

Resource Ordering

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

CS 245

Notes 10

28

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

CS 245

Notes 10

29

Timeout

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

CS 245

Notes 10

30

Wait-die

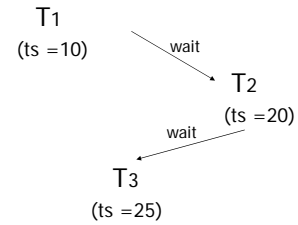
- Transactions given a timestamp when they arrive $ts(T_i)$
- T_i can only wait for T_j if $ts(T_i) < ts(T_j)$...else die

CS 245

Notes 10

31

Example:

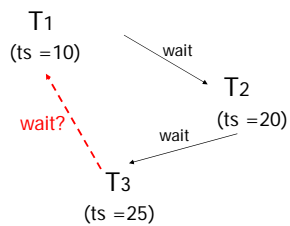


CS 245

Notes 10

32

Example:

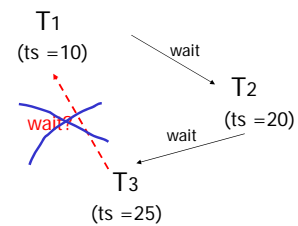


CS 245

Notes 10

33

Example:



CS 245

Notes 10

34

Starvation with Wait-Die

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

CS 245

Notes 10

35

Starvation with Wait-Die

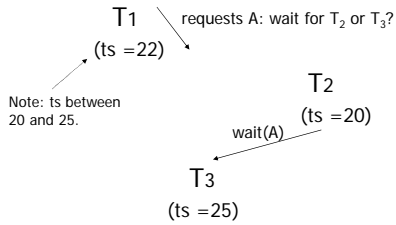
- Resubmit with original timestamp
- Guarantees no starvation
 - Transaction with oldest ts never dies
 - A transaction that dies will eventually have oldest ts and will complete...

CS 245

Notes 10

36

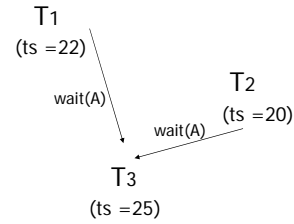
Second Example:



Note: ts between 20 and 25.

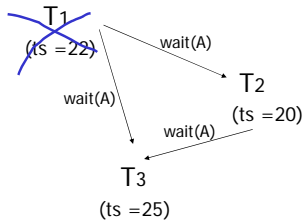
Second Example (continued):

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



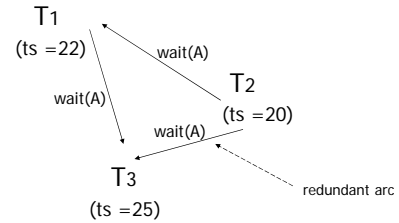
Second Example (continued):

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



Second Example (continued):

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

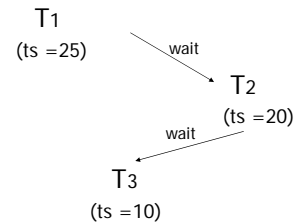


Wound-wait

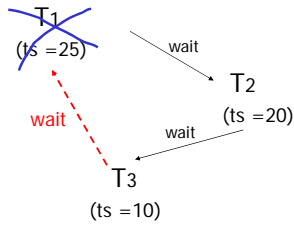
- Transactions given a timestamp when they arrive ... ts(T_i)
- T_i wounds T_j if ts(T_i) < ts(T_j)
 else T_i waits

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

Example:



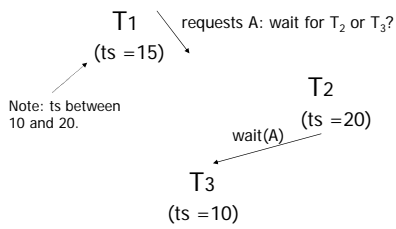
Example:



Starvation with Wound-Wait

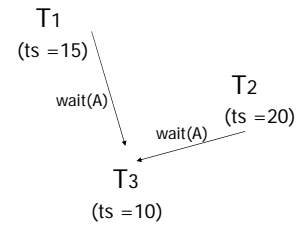
- When transaction dies, re-try later with what timestamp?
 - original timestamp
 - new timestamp (time of re-submit)

Second Example:



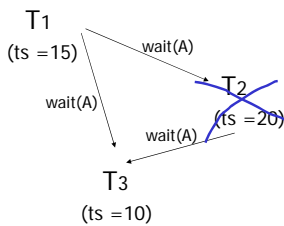
Second Example (continued):

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



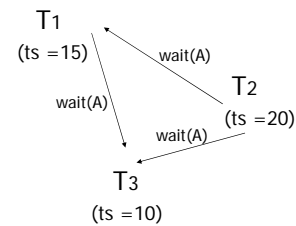
Second Example (continued):

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



Second Example (continued):

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



User/Program commands

Lots of variations, but in general

- Begin_work
- Commit_work
- Abort_work

CS 245

Notes 10

49

Nested transactions

User program:

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

CS 245

Notes 10

50

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
```

CS 245

Notes 10

51

Parallel Nested Transactions

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

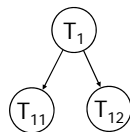
CS 245

Notes 10

52

Parallel Nested Transactions

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



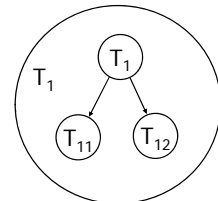
CS 245

Notes 10

53

Parallel Nested Transactions

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



CS 245


Notes 10

54

Locking

Locking

What are we really locking?



CS 245 Notes 10 55

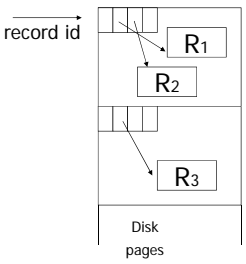
Example:

T_i :
 Read record r1
 :
 Read record r1
 :
 Modify record r3
 :
 :

do record locking

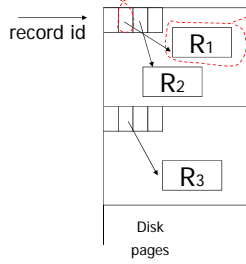
CS 245 Notes 10 56

But underneath:



CS 245 Notes 10 57

But underneath:



If we lock all data involved in read of R1, we may prevent an update to R2 (which may require reorganization within block)

CS 245 Notes 10 58

Solution: view DB at two levels

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

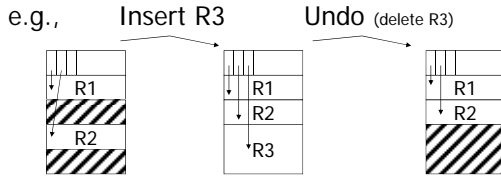
e.g., Insert record(X,Y,Z)
 Redo: insert(X,Y,Z)
 Undo: delete

CS 245 Notes 10 59

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

CS 245 Notes 10 60

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



CS 245

Notes 10

61

Logging Logical Actions

- Logical action typically span one block (physiological actions)
- Undo/redo log entry specifies undo/redo logical action

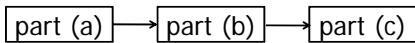
CS 245

Notes 10

62

Question

- How to deal with spanned record?



CS 245

Notes 10

63

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!

CS 245

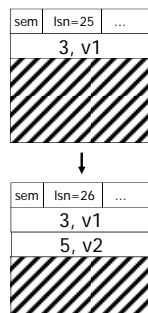
Notes 10

64

Solution: Add Log Sequence Number

Log record:

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

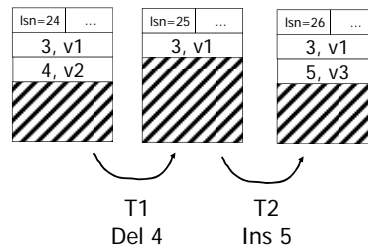


CS 245

Notes 10

65

Still Have a Problem!

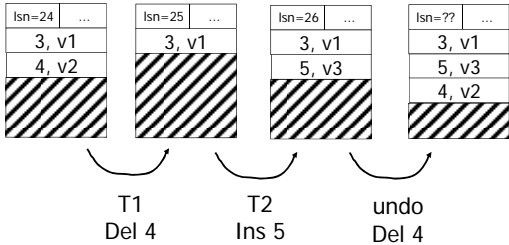


CS 245

Notes 10

66

Still Have a Problem!

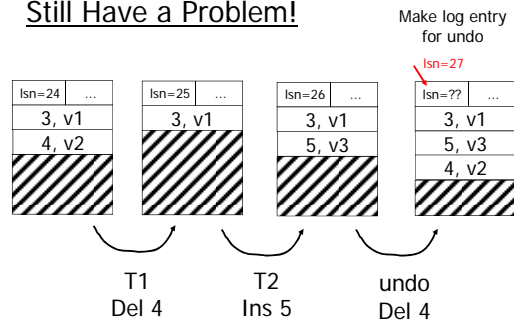


CS 245

Notes 10

67

Still Have a Problem!



CS 245

Notes 10

68

Compensation Log Records

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

CS 245

Notes 10

69

At Recovery: Example

Log:

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

CS 245

Notes 10

70

What to do with p2 (during T1 rollback)?

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

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

CS 245

Notes 10

71

Recovery Strategy

[1] Reconstruct state at time of crash

- Find latest valid checkpoint, ck , and let ac be its set of active transactions
- Scan log from ck to end:
 - For each log entry [Isn, page] do:
 - if $Isn(page) < Isn$ then redo action
 - If log entry is start or commit, update ac

CS 245

Notes 10

72

Recovery Strategy

[2] Abort uncommitted transactions

- Set ac contains transactions to abort
- Scan log from end to ck :
 - 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 ck and undo their actions

CS 245

Notes 10

73

Example: What To Do After Crash

Log:

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

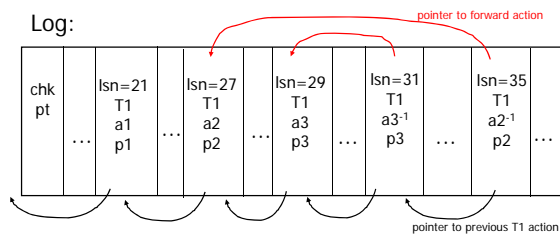
CS 245

Notes 10

74

During Undo: Skip Undo's

Log:



CS 245

Notes 10

75

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
 - $T_1, T_2, \dots, T_{n-1}, T_{n-1}^{-1}, T_{n-2}^{-1}, \dots, T_1^{-1}$
 - $T_1, T_2, \dots, T_{n-2}, T_{n-2}^{-1}, T_{n-3}^{-1}, \dots, T_1^{-1}$
 - ⋮
 - T_1, T_1^{-1}
 - nothing

CS 245

Notes 10

76

Summary

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

CS 245

Notes 10

77