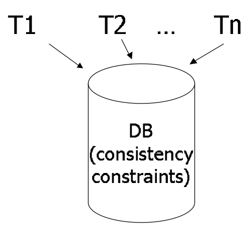
Chapter 18:
Concurrency Control

(Slides by Hector Garcia-Molina,
http://www-db.stanford.edu/~hector/cs245/notes.htm)

Chapter 18 Concurrency Control



Example:

T1:	Read(A)	T2:	Read(A)
	A ← A+100		A ← A×2
	Write(A)		Write(A)
	Read(B)		Read(B)
	B ← B+100		B ← B×2
	Write(B)		Write(B)

Constraint: A=B

Schedule A

T1	T2	A	B
		25	25
Read(A); A $\leftarrow$ A+100			
Write(A);		125	
Read(B); B $\leftarrow$ B+100;			
Write(B);			125
	Read(A);A $\leftarrow$ A$\times$2;		
	Write(A);	250	
	Read(B);B $\leftarrow$ B$\times$2;		
	Write(B);		250
		250	250

Schedule B

T1	T2	A	B
		25	25
	Read(A);A $\leftarrow$ A$\times$2;		
	Write(A);	50	
	Read(B);B $\leftarrow$ B$\times$2;		
	Write(B);		50
Read(A); A $\leftarrow$ A+100			
Write(A);		150	
Read(B); B $\leftarrow$ B+100;			
Write(B);			150
		150	150

Schedule C

T1	T2	A	B
		25	25
Read(A); A $\leftarrow$ A+100			
Write(A);		125	
	Read(A);A $\leftarrow$ A$\times$2;		
	Write(A);	250	
Read(B); B $\leftarrow$ B+100;			
Write(B);			125
	Read(B);B $\leftarrow$ B$\times$2;		
	Write(B);		250
		250	250

Schedule D

T1	T2	A	B
		25	25
Read(A); A ← A+100			
Write(A);		125	
	Read(A);A ← A×2;		
	Write(A);	250	
	Read(B);B ← B×2;		
	Write(B);		50
Read(B); B ← B+100;			
Write(B);			150
		250	150

Schedule E

Same as Schedule D but with new T2′

T1	T2′	A	B
		25	25
Read(A); A ← A+100			
Write(A);		125	
	Read(A);A ← A×1;		
	Write(A);	125	
	Read(B);B ← B×1;		
	Write(B);		25
Read(B); B ← B+100;			
Write(B);			125
		125	125

- Want schedules that are “good”, regardless of
  - initial state and
  - transaction semantics
- Only look at order of read and writes

Example:

$Sc=r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

Example:

$Sc = r_1(A)w_1(A)\underbrace{r_2(A)w_2(A)}\underbrace{r_1(B)w_1(B)}r_2(B)w_2(B)$

$Sc' = \underbrace{r_1(A)w_1(A)\ r_1(B)w_1(B)}_{T_1}\underbrace{r_2(A)w_2(A)r_2(B)w_2(B)}_{T_2}$

However, for Sd:

$Sd = r_1(A)\underbrace{w_1(A)}r_2(A)w_2(A)\ r_2(B)w_2(B)r_1(B)\underbrace{w_1(B)}$

- $T_2 \rightarrow T_1$
- Also, $T_1 \rightarrow T_2$

$T_1 \rightleftarrows T_2$

⇨ Sd cannot be rearranged into a serial schedule

⇨ Sd is not "equivalent" to any serial schedule

⇨ Sd is "bad"

Returning to Sc

$Sc = r_1(A) w_1(A) r_2(A) w_2(A) r_1(B) w_1(B) r_2(B) w_2(B)$

$T_1 \rightarrow T_2$ $\qquad\qquad$ $T_1 \rightarrow T_2$

☛ no cycles $\Rightarrow$ Sc is "equivalent" to a serial schedule (in this case T_1,T_2)

Chapter 18 13

Concepts

Transaction: sequence of $r_i(x)$, $w_i(x)$ actions

Conflicting actions: $r_1(A)$ / $w_2(A)$ $\quad$ $w_2(A)$ / $r_1(A)$ $\quad$ $w_1(A)$ / $w_2(A)$

Schedule: represents chronological order in which actions are executed

Serial schedule: no interleaving of actions or transactions

Chapter 18 14

Definition

S_1, S_2 are conflict equivalent schedules if S_1 can be transformed into S_2 by a series of swaps on non-conflicting actions.

Chapter 18 15

Definition

A schedule is conflict serializable if it is conflict equivalent to some serial schedule.

Precedence graph P(S) (S is schedule)

Nodes: transactions in S

Arcs: Ti $\rightarrow$ Tj whenever

- $p_i(A)$, $q_j(A)$ are actions in S
- $p_i(A) <_S q_j(A)$
- at least one of p_i, q_j is a write

Exercise:

- What is P(S) for
  S = $w_3(A)$ $w_2(C)$ $r_1(A)$ $w_1(B)$ $r_1(C)$ $w_2(A)$ $r_4(A)$ $w_4(D)$
- Is S serializable?

Lemma

S_1, S_2 conflict equivalent $\Rightarrow P(S_1)=P(S_2)$

Chapter 18 19

Note: $P(S_1)=P(S_2) \not\Rightarrow S_1, S_2$ conflict equivalent

Counter example:

$S_1=w_1(A)\ r_2(A) \quad w_2(B)\ r_1(B)$

$S_2=r_2(A)\ w_1(A) \quad r_1(B)\ w_2(B)$

Chapter 18 20

Theorem

$P(S_1)$ acyclic $\Longleftrightarrow S_1$ conflict serializable

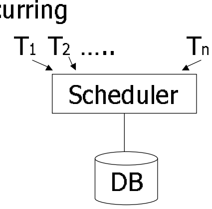
Chapter 18 21

How to enforce serializable schedules?

Option 1: run system, recording P(S); at end of day, check for P(S) cycles and declare if execution was good

How to enforce serializable schedules?

Option 2: prevent P(S) cycles from occurring

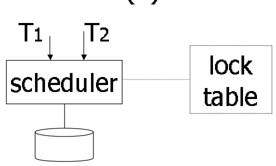


A locking protocol

Two new actions:

lock (exclusive): l_i (A)

unlock: u_i (A)



Rule #1: Consistent transactions

T_i: … $l_i(A)$ … $p_i(A)$ … $u_i(A)$ …

Chapter 18 25

Rule #2 Legal scheduler

S = …….. $l_i(A)$ ………… $u_i(A)$ ………

$\underleftrightarrow{\quad\quad}$
no $l_j(A)$

Chapter 18 26

Exercise:

- What schedules are legal?
  What transactions are consistent?

  S1 = $l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
  $r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

  S2 = $l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)$
  $l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$

  S3 = $l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B)$
  $l_2(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

Chapter 18 27

Schedule F

T1	T2
$l_1(A)$;Read(A)	
$A \leftarrow A+100$;Write(A);$u_1(A)$	
	$l_2(A)$;Read(A)
	$A \leftarrow Ax2$;Write(A);$u_2(A)$
	$l_2(B)$;Read(B)
	$B \leftarrow Bx2$;Write(B);$u_2(B)$
$l_1(B)$;Read(B)	
$B \leftarrow B+100$;Write(B);$u_1(B)$	

Schedule F

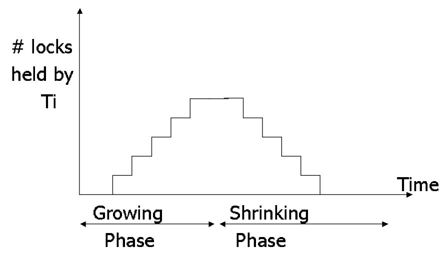
T1	T2	A	B
		25	25
$l_1(A)$;Read(A)			
$A \leftarrow A+100$;Write(A);$u_1(A)$		125	
	$l_2(A)$;Read(A)		
	$A \leftarrow Ax2$;Write(A);$u_2(A)$	250	
	$l_2(B)$;Read(B)		
	$B \leftarrow Bx2$;Write(B);$u_2(B)$		50
$l_1(B)$;Read(B)			
$B \leftarrow B+100$;Write(B);$u_1(B)$			150
		250	150

Rule #3 Two phase locking (2PL)

for transactions

T_i = ……. $l_i(A)$ ………… $u_i(A)$ ………

no unlocks   no locks



Schedule G

T1	T2
l_1(A);Read(A)	
A ← A+100;Write(A)	
l_1(B); u_1(A)	
	l_2(A);Read(A)
	A ← Ax2;Write(A);l_2(B) delayed

Schedule G

T1	T2
l_1(A);Read(A)	
A ← A+100;Write(A)	
l_1(B); u_1(A)	
	l_2(A);Read(A)
	A ← Ax2;Write(A);l_2(B) delayed
Read(B);B ← B+100	
Write(B); u_1(B)	

Schedule G

T1	T2
$l_1(A)$;Read(A)	
A ← A+100;Write(A)	
$l_1(B)$; $u_1(A)$	
	$l_2(A)$;Read(A)
	A ← Ax2;Write(A);$l_2(B)$ delayed
Read(B);B ← B+100	
Write(B); $u_1(B)$	
	$l_2(B)$; $u_2(A)$;Read(B)
	B ← Bx2;Write(B);$u_2(B)$;

Chapter 18 34

Schedule H (T2 reversed)

T1	T2
$l_1(A)$; Read(A)	$l_2(B)$;Read(B)
A ← A+100;Write(A)	B ← Bx2;Write(B)
$l_1(B)$ delayed	$l_2(A)$ delayed

deadlock

Chapter 18 35

- Assume deadlocked transactions are rolled back
  - They have no effect
  - They do not appear in schedule

E.g., Schedule H on previous slide

Chapter 18 36

We can show that
rules #1,2,3 $\Rightarrow$ conflict-serializable schedules

(see textbook)

- 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

Shared locks

So far:

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

Do not conflict

Instead:

$S = \ldots ls_1(A)\ r_1(A)\ ls_2(A)\ r_2(A) \ldots.\ us_1(A)\ us_2(A)$

Lock actions

l-t_i(A): lock A in t mode (t is S or X)

u-t_i(A): unlock t mode (t is S or X)

Shorthand:

u_i(A): unlock whatever modes T_i has locked A

- What about transactions that read and write same object?

Option 1: Request exclusive lock

T_i = …l-X_1(A) … r_1(A) … w_1(A) … u_1(A) …

- What about transactions that read and write same object?

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) …u1(A)…

The same as:
- Getting 2nd lock on A, or
- Dropping S, getting X lock

Compatibility matrix

Comp

	S	X
S	true	false
X	false	false

Lock types beyond S/X

Examples:

(1) increment lock
(see textbook)

(2) update lock

Upgrate locks

Common deadlock problem with upgrades:

T1	T2
$l\text{-}S_1(A)$	
	$l\text{-}S_2(A)$
$l\text{-}X_1(A)$	
	$l\text{-}X_2(A)$

--- Deadlock ---

Solution

If T_i wants to read A and knows it may later want to write A, it requests update lock (not shared)

How does locking work in practice?

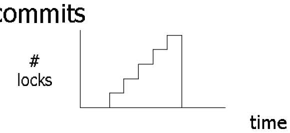
- Every system is different

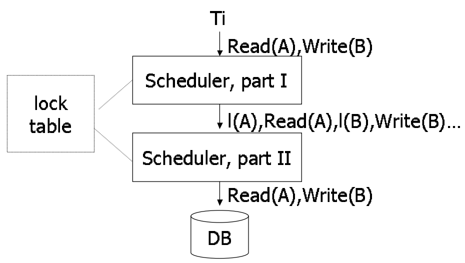
  (E.g., may not even provide CONFLICT-SERIALIZABLE schedules)
- But here is one (simplified) way ...

Sample Locking System:

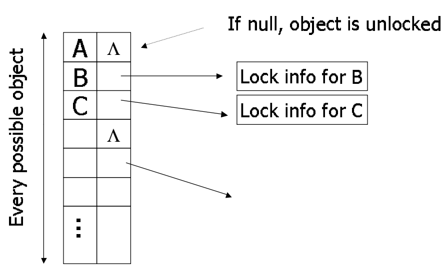
(1) Don't trust transactions to request/release locks

(2) Hold all locks until transaction commits

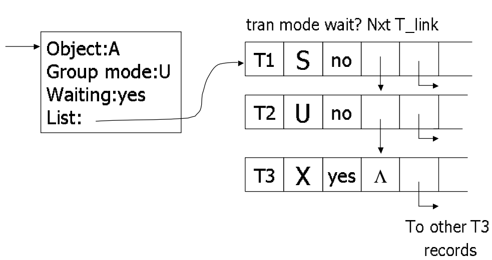




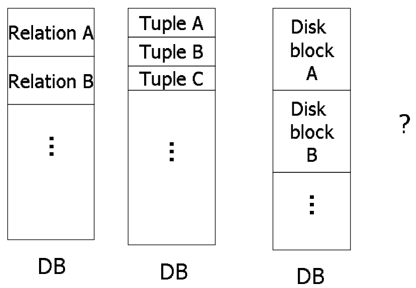
Lock table Conceptually



Lock info for A - example



What are the objects we lock?



- Locking works in any case, but should we choose small or large objects?
- If we lock large objects (e.g., relations)
  - Need few locks
  - Low concurrency
- If we lock small objects (e.g., tuples,fields)
  - Need more locks
  - More concurrency

Summary

Have looked at 2PL, a concurrency-control mechanism commonly used in practice

Others (in the textbook):
- Multiple granularity
- Tree (index) protocols
- Timestamping
- Validation