Introduction to Database Systems CSE 444

Lectures 15-16: Recovery

May 7-9, 2008

Announcements

Homework 3:

- Attributes v.s. elements: /item v.s. /@item
- Data is not clean
  - OK to return any sensible answer, no need to clean
- See the two examples in the mini-tutorial (e.g fn:string)
- Check the lecture notes (e.g. for group-by)
- If query doesn't work, try a simpler one to debug

Outline

- Undo logging 17.2
- Redo logging 17.3
- Redo/undo 17.4

Transaction Management

Two parts:

- Recovery from crashes: ACID
- Concurrency control: ACID

Both operate on the buffer pool

Recovery

From which of the events below can a database actually recover ?

- Wrong data entry
- Disk failure
- Fire / earthquake / bankrupcy / ….
- Systems crashes

Recovery

Type of Crash	Prevention
Wrong data entry	Constraints and Data cleaning
Disk crashes	Redundancy: e.g. RAID, archive
Fire, theft, bankruptcy…	Buy insurance, Change jobs…
System failures: e.g. power	DATABASE RECOVERY

Most frequent

System Failures

- Each transaction has *internal state*
- When system crashes, internal state is lost
  - Don't know which parts executed and which didn't
- Remedy: use a **log**
  - A file that records every single action of the transaction

Transactions

- Assumption: the database is composed of ***elements***
  - Usually 1 element = 1 block
  - Can be smaller (=1 record) or larger (=1 relation)
- Assumption: each transaction reads/writes some elements

Primitive Operations of Transactions

- READ(X,t)
  - copy element X to transaction local variable t
- WRITE(X,t)
  - copy transaction local variable t to element X
- INPUT(X)
  - read element X to memory buffer
- OUTPUT(X)
  - write element X to disk

Example

```
START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t)
COMMIT;
```

Atomicity:
BOTH A and B
are multiplied by 2

```
READ(A,t); t := t*2; WRITE(A,t);
READ(B,t); t := t*2; WRITE(B,t)
```

	Transaction	Buffer pool		Disk	
Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Crash !

Crash occurs after OUTPUT(A), before OUTPUT(B)
We lose atomicity

The Log

- An append-only file containing log records
- Note: multiple transactions run concurrently, log records are interleaved
- After a system crash, use log to:
  - Redo some transaction that didn't commit
  - Undo other transactions that didn't commit
- Three kinds of logs: undo, redo, undo/redo

Undo Logging

Log records

- <START T>
  - transaction T has begun
- <COMMIT T>
  - T has committed
- <ABORT T>
  - T has aborted
- <T,X,v>
  - T has updated element X, and its *old* value was v

Action	t	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
INPUT(A)		8		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>

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
INPUT(A)		8		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>

Crash !

WHAT DO WE DO ?

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
INPUT(A)		8		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>

WHAT DO WE DO ?

After Crash

- In the first example:
  - We UNDO both changes: A=8, B=8
  - The transaction is atomic, since none of its actions has been executed
- In the second example
  - We don't undo anything
  - The transaction is atomic, since both it's actions have been executed

18

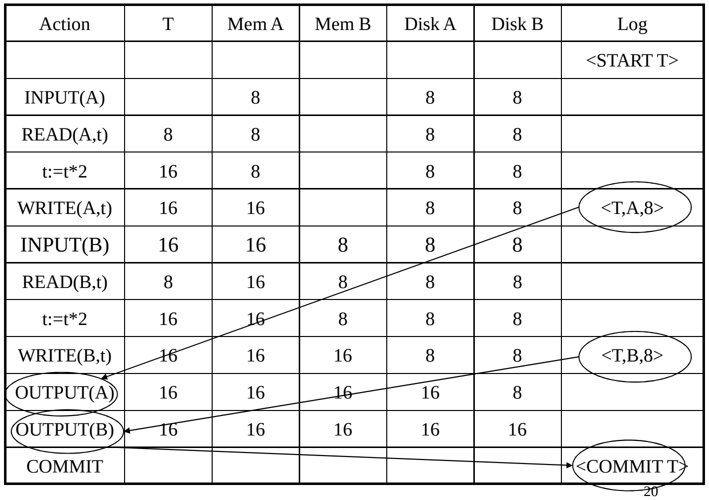
Undo-Logging Rules

U1: If T modifies X, then <T,X,v> must be written to disk before OUTPUT(X)

U2: If T commits, then OUTPUT(X) must be written to disk before <COMMIT T>

- Hence: OUTPUTs are done *early*, before the transaction commits

19



Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
INPUT(A)		8		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>

20

Recovery with Undo Log

After system's crash, run recovery manager

- Idea 1. Decide for each transaction T whether it is completed or not
  - <START T>….<COMMIT T>…. = yes
  - <START T>….<ABORT T>……. = yes
  - <START T>……………………… = no
- Idea 2. Undo all modifications by incomplete transactions

Recovery with Undo Log

Recovery manager:

- Read log <u>from the end</u>; cases:

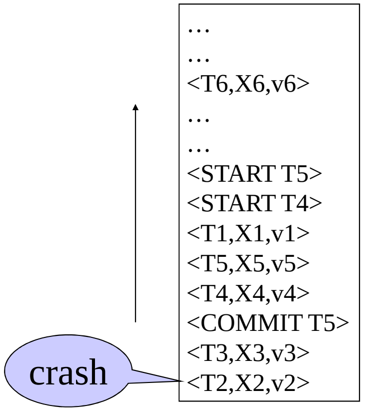
  <COMMIT T>: mark T as completed

  <ABORT T>: mark T as completed

  <T,X,v>: if T is not completed
  then write X=v to disk
  else ignore

  <START T>: ignore

Recovery with Undo Log



```
…
…
<T6,X6,v6>
…
…
<START T5>
<START T4>
<T1,X1,v1>
<T5,X5,v5>
<T4,X4,v4>
<COMMIT T5>
<T3,X3,v3>
<T2,X2,v2>
```

Question1 in class:
Which updates are
undone ?

Question 2 in class:
How far back
do we need to
read in the log ?

Recovery with Undo Log

- Note: all undo commands are *<u>idempotent</u>*
  - If we perform them a second time, no harm is done
  - E.g. if there is a system crash during recovery, simply restart recovery from scratch

Recovery with Undo Log

When do we stop reading the log ?

- We cannot stop until we reach the beginning of the log file
- This is impractical

Instead: use checkpointing

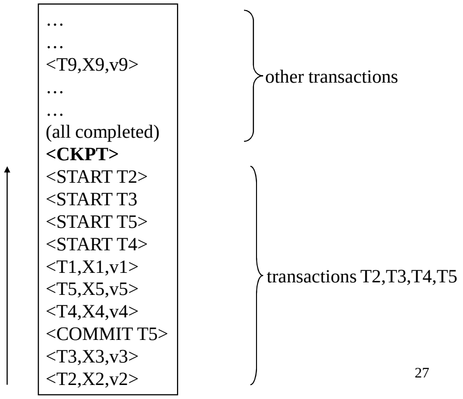
Checkpointing

Checkpoint the database periodically

- Stop accepting new transactions
- Wait until all current transactions complete
- Flush log to disk
- Write a <CKPT> log record, flush
- Resume transactions

Undo Recovery with Checkpointing

During recovery, can stop at first <CKPT>



…
…
<T9,X9,v9>
…
…
(all completed)
<CKPT>
<START T2>
<START T3
<START T5>
<START T4>
<T1,X1,v1>
<T5,X5,v5>
<T4,X4,v4>
<COMMIT T5>
<T3,X3,v3>
<T2,X2,v2>

Nonquiescent Checkpointing

- Problem with checkpointing: database freezes during checkpoint
- Would like to checkpoint while database is operational
- Idea: nonquiescent checkpointing

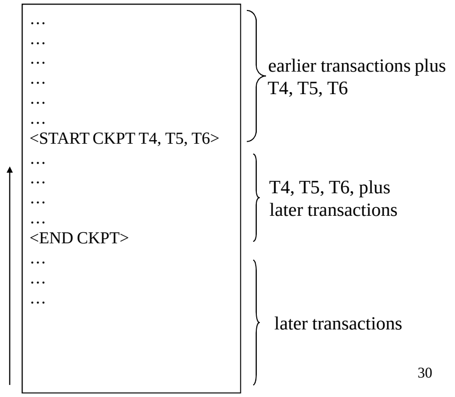
Quiescent = being quiet, still, or at rest; inactive
Non-quiescent = allowing transactions to be active

Nonquiescent Checkpointing

- Write a <START CKPT(T1,…,Tk)> where T1,…,Tk are all active transactions
- Continue normal operation
- When all of T1,…,Tk have completed, write <END CKPT>

Undo Recovery with Nonquiescent Checkpointing

During recovery, can stop where?


…
…
…
…
…
…
<START CKPT T4, T5, T6>
…
…
…
…
<END CKPT>
…
…
…
earlier transactions plus T4, T5, T6
T4, T5, T6, plus later transactions
later transactions

Q: why do we need <END CKPT> ?

Redo Logging

Log records

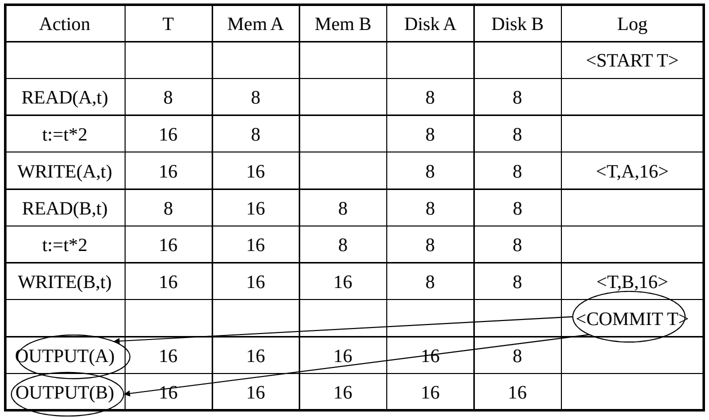
- <START T> = transaction T has begun
- <COMMIT T> = T has committed
- <ABORT T>= T has aborted
- <T,X,v>= T has updated element X, and its *new* value is v

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

Redo-Logging Rules

R1: If T modifies X, then both <T,X,v> and <COMMIT T> must be written to disk before OUTPUT(X)

- Hence: OUTPUTs are done *late*



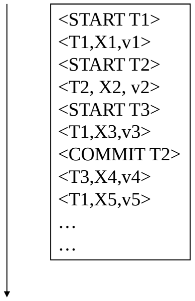
Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,16>
						<COMMIT T>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

Recovery with Redo Log

After system’s crash, run recovery manager

- Step 1. Decide for each transaction T whether it is completed or not
  - <START T>….<COMMIT T>….   = yes
  - <START T>….<ABORT T>…….   = yes
  - <START T>………………………   = no
- Step 2. Read log from the beginning, redo all updates of *committed* transactions

Recovery with Redo Log



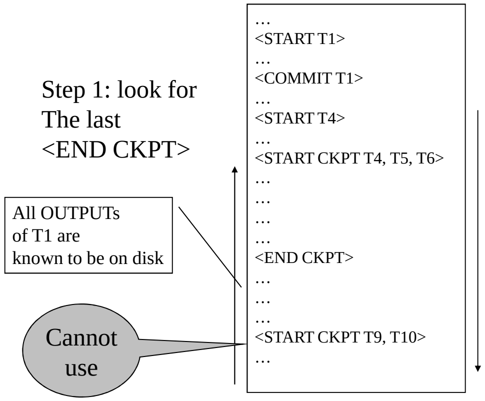
<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
…
…

Nonquiescent Checkpointing

- Write a <START CKPT(T1,…,Tk)> where T1,…,Tk are all active transactions
- Flush to disk all blocks of committed transactions (*dirty blocks*), while continuing normal operation
- When all blocks have been written, write <END CKPT>

37

Redo Recovery with Nonquiescent Checkpointing



Step 1: look for The last <END CKPT>

All OUTPUTs of T1 are known to be on disk

Cannot use

…
<START T1>
…
<COMMIT T1>
…
<START T4>
…
<START CKPT T4, T5, T6>
…
…
…
…
<END CKPT>
…
…
…
<START CKPT T9, T10>
…

Step 2: redo from the earliest start of T4, T5, T6 ignoring transactions committed earlier

38

Comparison Undo/Redo

- Undo logging:
  - OUTPUT must be done early
  - If <COMMIT T> is seen, T definitely has written all its data to disk (hence, don't need to redo) – inefficient
- Redo logging
  - OUTPUT must be done late
  - If <COMMIT T> is not seen, T definitely has not written any of its data to disk (hence there is not dirty data on disk, no need to undo) – inflexible
- Would like more flexibility on when to OUTPUT: undo/redo logging (next)

39

Undo/Redo Logging

Log records, only one change

- <T,X,u,v>= T has updated element X, its *old* value was u, and its *new* value is v

40

Undo/Redo-Logging Rule

UR1: If T modifies X, then <T,X,u,v> must be written to disk before OUTPUT(X)

Note: we are free to OUTPUT early or late relative to <COMMIT T>

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

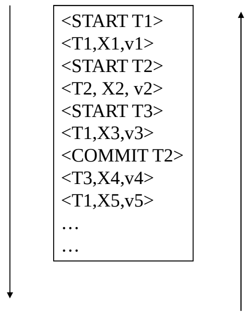
Can OUTPUT whenever we want: before/after COMMIT

Recovery with Undo/Redo Log

After system's crash, run recovery manager

- Redo all committed transaction, top-down
- Undo all uncommitted transactions, bottom-up

Recovery with Undo/Redo Log



<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
…
…