

CIS 560 – Database System Concepts

Lecture 19

Transaction Management: Recovery

October 18, 2013

Credits for slides: Chang, Ullman, Whitehead.

Copyright: Caragea, 2013.

Announcements

- HW5 - SQL/JDBC assignment due tomorrow night (one da)
- HW6 will be posted tonight

Outline

Last:

- Buffer management 15.7

Today:

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

Next:

- Serial and serializable schedules 18.1
- Conflict serializability 18.2
- Locks 18.3

3

Undo Logging

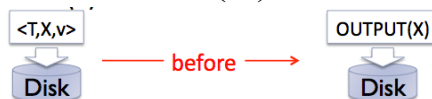
Log records

- <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

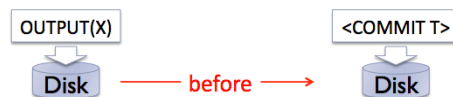
4

Undo-Logging Rules

U1: If T modifies X, then $\langle T, X, v \rangle$ must be written to disk before OUTPUT(X)



U2: If T commits, then OUTPUT(X) must be written to disk before $\langle \text{COMMIT } T \rangle$



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

5

Example

Given this undo log, when can each data item be output to disk?

- A: after 2
- B: after 3
- C: after 5, before 12
- D: after 7
- E: after 8, before 12
- F: after 10
- G: after 11

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<T2, E, e>
9	<START T4>
10	<T4, F, f>
11	<T3, G, g>
12	<COMMIT T2>

6

Recovery with Undo Log

After system's crash, run recovery manager

- 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
- 2. Undo all modifications by incomplete transactions

7

Recovery with Undo Log

Recovery manager:

- Read log from the end; 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

After all changes are made, write <ABORT T> to the log, for each transaction T that was incomplete and then flush the log.

8

Example

After writing these log entries, the DBMS crashes. What does it do when it restarts?

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<T2, E, e>
9	<START T4>
10	<T4, F, f>
11	<T3, G, g>
12	<COMMIT T2>

9

Example

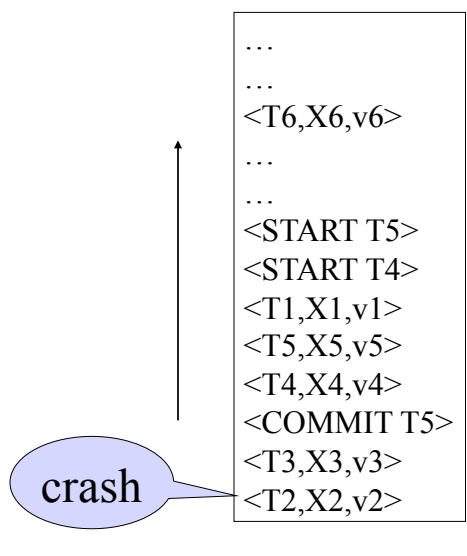
After writing these log entries, the DBMS crashes. What does it do when it restarts?

- Scan for transactions to undo: T1, T3, T4
- G, F, D, B, A reverted (in that order)
- <ABORT> written for T1, T3, T4

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<T2, E, e>
9	<START T4>
10	<T4, F, f>
11	<T3, G, g>
12	<COMMIT T2>

10

Recovery with Undo Log



The diagram shows a vertical list of log entries. From bottom to top, they are: <T2,X2,v2>, <T3,X3,v3>, <COMMIT T5>, <T4,X4,v4>, <T5,X5,v5>, <T1,X1,v1>, <START T4>, <START T5>, and <T6,X6,v6>. Above <T6,X6,v6> are three dots. A blue speech bubble labeled "crash" points to the entry <T2,X2,v2>. An upward-pointing arrow is positioned to the left of the log entries, starting from the level of the crash and extending upwards.

Question1: Which updates are undone ?

Question 2: What happens if there is a second crash, during recovery ?

Question 3: How far back do we need to read in the log ?

11

Recovery with Undo Log

- Note: all undo commands are idempotent
 - 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

13

Checkpointing

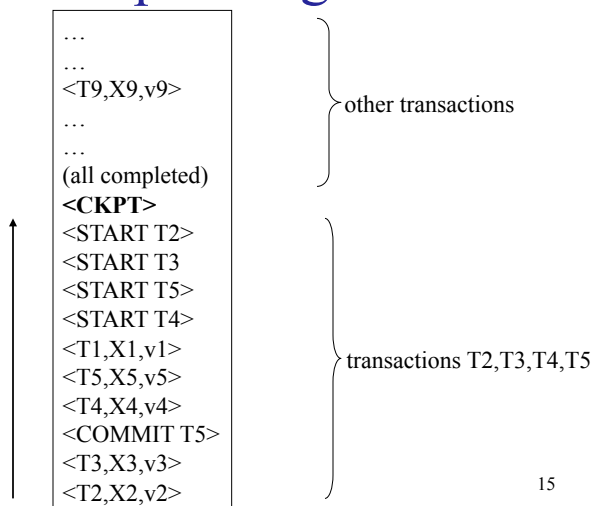
Checkpoint the database periodically

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

14

Undo Recovery with Checkpointing

During recovery,
can stop at first
<CKPT>



15

Nonquiescent Checkpointing

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

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

16

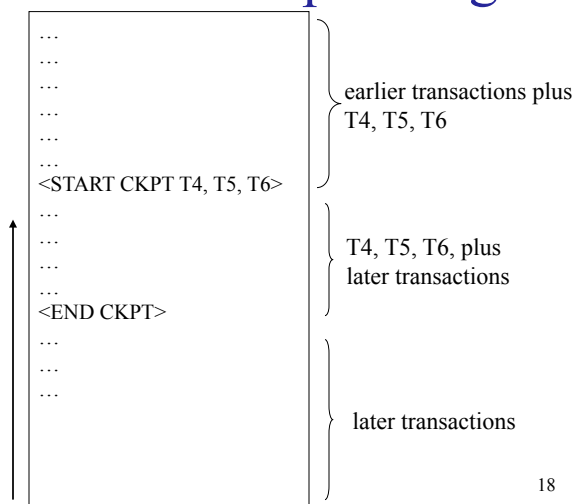
Nonquiescent Checkpointing

- Write a $\langle \text{START CKPT}(T_1, \dots, T_k) \rangle$ where T_1, \dots, T_k are all active transactions
- Continue normal operation
- When all of T_1, \dots, T_k have completed, write $\langle \text{END CKPT} \rangle$

17

Undo Recovery with Nonquiescent Checkpointing

During recovery,
can stop at first
 $\langle \text{CKPT} \rangle$



Q: do we need
 $\langle \text{END CKPT} \rangle$?

18

Example

The DBMS crashes with this undo log. What do we do to recover?

- Which log entries are read?
- Which transactions are undone?
- Which data do we change?

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<COMMIT T1>
9	<START CKPT (T2, T3)>
10	<T2, E, e>
11	<START T4>
12	<T4, F, f>
13	<T3, G, g>
14	<COMMIT T3>
15	<COMMIT T2>
16	<END CKPT>
17	<COMMIT T4>

Example

The DBMS crashes with this undo log. What do we do to recover?

- Which log entries are read?
- Which transactions are undone?

From end to 9: <START CKPT>

None; all have committed

- Which data do we change?

None; no transactions to undo

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<COMMIT T1>
9	<START CKPT (T2, T3)>
10	<T2, E, e>
11	<START T4>
12	<T4, F, f>
13	<T3, G, g>
14	<COMMIT T3>
15	<COMMIT T2>
16	<END CKPT>
17	<COMMIT T4>

Redo Logging

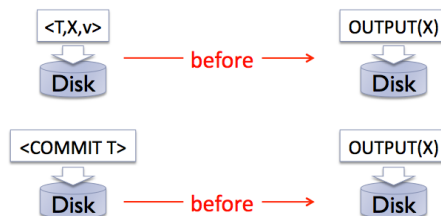
Log records

- $\langle \text{START } T \rangle$ = transaction T has begun
- $\langle \text{COMMIT } T \rangle$ = T has committed
- $\langle \text{ABORT } T \rangle$ = T has aborted
- $\langle T, X, v \rangle$ = T has updated element X , and its new value is v

21

Redo-Logging Rules

R1: If T modifies X , then both $\langle T, X, v \rangle$ and $\langle \text{COMMIT } T \rangle$ must be written to disk before $\text{OUTPUT}(X)$



- Hence: OUTPUTs are done late

22

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	

23

Example

- When can we output each item?

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<T2, E, e>
9	<START T4>
10	<T4, F, f>
11	<T3, G, g>
12	<COMMIT T2>

24

Example

- When can we output each item?

C, E: after 12

Others: can't (given the log available)

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<T2, E, e>
9	<START T4>
10	<T4, F, f>
11	<T3, G, g>
12	<COMMIT T2>

25

Recovery with Redo Log

After system's crash, run recovery manager

- Step 1. Decide for each transaction T whether we need to redo or not
 - <START T>....<COMMIT T>.... = yes
 - <START T>....<ABORT T>..... = no
 - <START T>..... = no
- Step 2. Read log from the beginning, redo all updates of committed transactions
- Step 3. For each incomplete transaction T, write an <ABORT T> to the log and flush the log

26

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>
 ...
 ...

27

Example

- How can we recover from this redo log?

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<T2, E, e>
9	<START T4>
10	<T4, F, f>
11	<T3, G, g>
12	<COMMIT T2>

28

Example

- How can we recover from this redo log?

Scan for transactions
to redo: only T2
C and E rewritten

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<T2, E, e>
9	<START T4>
10	<T4, F, f>
11	<T3, G, g>
12	<COMMIT T2>

29

Nonquiescent Checkpointing

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

30

Redo Recovery with Nonquiescent Checkpointing

Step 1: look for the last <END CKPT>

All OUTPUTs of T1 are guaranteed 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

31

Which is undo/redo?

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<COMMIT T1>
9	<START CKPT (T2, T3)>
10	<T2, E, e>
11	<START T4>
12	<T4, F, f>
13	<T3, G, g>
14	<COMMIT T3>
15	<END CKPT>
16	<COMMIT T2>
17	<COMMIT T4>

1	<START T1>
2	<T1, A, a>
3	<T1, B, b>
4	<START T2>
5	<T2, C, c>
6	<START T3>
7	<T3, D, d>
8	<COMMIT T1>
9	<START CKPT (T2, T3)>
10	<T2, E, e>
11	<START T4>
12	<T4, F, f>
13	<T3, G, g>
14	<COMMIT T3>
15	<COMMIT T2>
16	<END CKPT>
17	<COMMIT T4>

32

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)

33

Undo/Redo Logging

Log records, only one change

- $\langle T, X, u, v \rangle = T$ has updated element X , its old value was u , and its new value is v

34

Undo/Redo-Logging Rule

UR1: If T modifies X, then $\langle T, X, u, v \rangle$ must be written to disk before OUTPUT(X)

Note: we are free to OUTPUT early or late relative to $\langle \text{COMMIT } T \rangle$

35

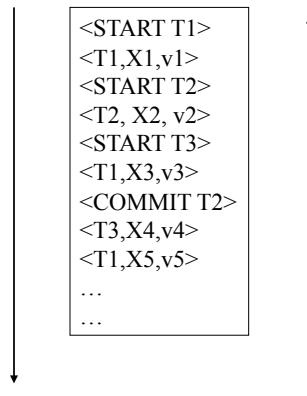
Action	T	Mem A	Mem B	Disk A	Disk B	Log
						$\langle \text{START } T \rangle$
READ(A,t)	8	8		8	8	
$t := t * 2$	16	8		8	8	
WRITE(A,t)	16	16		8	8	$\langle T, A, 8, 16 \rangle$
READ(B,t)	8	16	8	8	8	
$t := t * 2$	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	$\langle T, B, 8, 16 \rangle$
OUTPUT(A)	16	16	16	16	8	
						$\langle \text{COMMIT } T \rangle$
OUTPUT(B)	16	16	16	16	16	

Can OUTPUT whenever we want: before/after COMMIT³⁶

Recovery with Undo/Redo Log

After system's crash, run recovery manager

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



37

Undo-redo checkpointing

<CKPT(...)> denotes the beginning of a checkpoint and lists the currently active transactions.

<END CKPT> is written to disk once *all dirty pages* have been flushed to disk (all data prior to start). The redo phase precedes the undo phase during the recovery.

Which are the transactions whose actions the recovery manager needs to redo?

Which are the transactions whose actions the recovery manager needs to undo?

Indicate the actions of the recovery manager on all the elements, separately during the Redo and the Undo phase.

```

<START T1>
<T1 A 1 2>
<START T2>
<COMMIT T1>
<START T3>
<T3 A 2 3>
<START T4>
<CKPT(T2,T3,T4)>
<T2 B 10 20>
<COMMIT T2>
<START T5>
<T5 D 1000 2000>
<T4 C 100 200>
<COMMIT T5>
<START T6>
<END CKPT>
<T6 D 2000 3000>

```

Undo-redo recovery log

Which are the transactions whose actions the recovery manager needs to redo?

T2, T5

Which are the transactions whose actions the recovery manager needs to undo?

T3, T4, T6

Indicate the actions of the recovery manager on all the elements, separately during the Redo and the Undo phase.

Redo B=20, D=2000
Undo A= 2, D=2000, C=100

```
<START T1>
<T1 A 1 2>
<START T2>
<COMMIT T1>
<START T3>
<T3 A 2 3>
<START T4>
<CKPT(T2,T3,T4)>
<T2 B 10 20>
<COMMIT T2>
<START T5>
<T5 D 1000 2000>
<T4 C 100 200>
<COMMIT T5>
<START T6>
<END CKPT>
<T6 D 2000 3000>
```

Granularity of the Log

- Physical logging: element = physical page
- Logical logging: element = data record
- What are the pros and cons?

Granularity of the Log

Modern DBMS:

- Physical logging for the REDO part
 - Efficiency
- Logical logging for the UNDO part
 - For ROLLBACKs

41