

CSE 444: Database Internals

Section 8:

Transactions - Recovery

Review in this section

- 1. UNDO logging
- 1. REDO logging
- 1. Updating ARIES Data Structures

Undo Logging

- Two Rules:
 - 1. If a transaction writes element **X**, then the log record of this update **<T,X,v>** must be written to disk before the new value of **X** is written to disk.
 - 2. If a transaction commits, then the **COMMIT** must be written to disk only after all elements changed by the transaction have been written to disk.

UNDO LOG RULES

1. $\langle T, X, v \rangle$ before $\text{OUTPUT}(X)$
2. $\text{OUTPUT}(X)$ before $\langle \text{COMMIT} \rangle$

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

When recovering (with UNDO logging)...

- We can not simply ignore the log before a recent commit
 - Many transactions interleave at once. If we truncate before a commit for a transaction, any information about those unfinished transactions would be lost.
- Instead, we can use checkpoint the log periodically...

Review: Checkpointing

- **Checkpointing (naïve)**

- Write a $\langle \text{START CKPT}(T_1, \dots, T_k) \rangle$. Flush log to disk
- **Stop accepting new transactions**
- Wait until all active transactions abort/commit
- Write $\langle \text{CKPT} \rangle$. Flush log to disk.
- Resume accepting transactions

- **Nonquiescent Checkpointing**

- Write a $\langle \text{START CKPT}(T_1, \dots, T_k) \rangle$. Flush log to disk
- Continue normal operation
- When all of $T_1, \dots, T_k$ have completed, write $\langle \text{END CKPT} \rangle$. Flush log to disk
- **More efficient, system does not seem to be stalled**

Problem 1.

UNDO Logging

```
LSN1      <START  T1>
LSN2      <T1  X  5>
LSN3      <START  T2>
LSN4      <T1  Y  7>
LSN5      <T2  X  9>
LSN6      <START  T3>
LSN7      <T3  Z 11>
LSN8      <COMMIT T1>
LSN9      <START  CKPT(T2,T3)>
LSN10     <T2  X 13>
LSN11     <T3  Y 15>
```

CRASH

1.

Show how far back in the recovery manager needs to read the log

(which LSN do we need to read up to?)

UNDO: How far to scan log from the end?

- **Case 1:** See **<END CKPT>** first
 - All incomplete transactions began after <START CKPT...>
- **Case 2:** See **<START CKPT(T1..TK)>** first
 - Incomplete transactions began after <START CKPT...> or incomplete ones among T1..TK
 - Find the earliest <START Ti> among them
 - At most we have to go until the previous <START CKPT>...<END CKPT>

LSN1	<START T1>
LSN2	<T1 X 5>
LSN3	<START T2>
LSN4	<T1 Y 7>
LSN5	<T2 X 9>
LSN6	<START T3>
LSN7	<T3 Z 11>
LSN8	<COMMIT T1>
LSN9	<START CKPT(T2,T3)>
LSN10	<T2 X 13>
LSN11	<T3 Y 15>

CRASH

Problem 1.

UNDO Logging

```
LSN1      <START  T1>
LSN2      <T1  X  5>
LSN3      <START  T2>
LSN4      <T1  Y  7>
LSN5      <T2  X  9>
LSN6      <START  T3>
LSN7      <T3  Z 11>
LSN8      <COMMIT T1>
LSN9      <START  CKPT(T2,T3)>
LSN10     <T2  X 13>
LSN11     <T3  Y 15>
          *CRASH*
```

1.

Show how far back in the recovery manager needs to read the log

(write the earliest LSN)

LSN3

(start of the earliest transaction among incomplete transactions)

Problem 1. UNDO Logging

```
LSN1      <START  T1>
LSN2      <T1  X  5>
LSN3      <START  T2>
LSN4      <T1  Y  7>
LSN5      <T2  X  9>
LSN6      <START  T3>
LSN7      <T3  Z 11>
LSN8      <COMMIT T1>
LSN9      <START  CKPT(T2,T3)>
LSN10     <T2  X 13>
LSN11     <T3  Y 15>
```

CRASH

2.

Show the actions of the recovery manager during recovery.

Problem 1. UNDO Logging

```
LSN1      <START T1>
LSN2      <T1 X 5>
LSN3      <START T2>
LSN4      <T1 Y 7>
LSN5      <T2 X 9>
LSN6      <START T3>
LSN7      <T3 Z 11>
LSN8      <COMMIT T1>
LSN9      <START CKPT(T2,T3)>
LSN10     <T2 X 13>
LSN11     <T3 Y 15>
          *CRASH*
```

2.

Show the actions of the recovery manager during recovery.

Y = 15

X = 13

Z = 11

X = 9

Redo Logging

- One Rule:
 - 1. Before modifying any element X on disk, all log records pertaining to this modification ($\langle T, X, v \rangle$ and the $\langle \text{COMMIT } T \rangle$), must appear on disk.

Action	T	<u>REDO LOG RULE</u> Both <T, X, v> and <COMMIT > before OUTPUT(X) v = new value				Log
						<START T>
READ(A,t)	8				8	
t:=t*2	16				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	

Problem 2: REDO Logging

- | | |
|------------------------|-------------------------|
| 1. < START T1 > | 11. < T2, E, 5 > |
| 2. < T1, A, 10 > | 12. < COMMIT T2 > |
| 3. < START T2 > | 13. < T3, F, 1 > |
| 4. < T2, B, 5 > | 14. < T4, G, 15 > |
| 5. < T1, C, 7 > | 15. < END CKPT > |
| 6. < START T3 > | 16. < COMMIT T3 > |
| 7. < T3, D, 12 > | 17. < START T5 > |
| 8. < COMMIT T1 > | 18. < T5, H, 3 > |
| 9. < START CKPT ???? > | 19. < START CKPT ???? > |
| 10. < START T4 > | 20. < COMMIT T5 > |
| | * CRASH * |

1.
What are the
correct values of
the two
<START CKPT ????>
records?

Problem 2: REDO Logging

- | | |
|------------------------|-------------------------|
| 1. < START T1 > | 11. < T2, E, 5 > |
| 2. < T1, A, 10 > | 12. < COMMIT T2 > |
| 3. < START T2 > | 13. < T3, F, 1 > |
| 4. < T2, B, 5 > | 14. < T4, G, 15 > |
| 5. < T1, C, 7 > | 15. < END CKPT > |
| 6. < START T3 > | 16. < COMMIT T3 > |
| 7. < T3, D, 12 > | 17. < START T5 > |
| 8. < COMMIT T1 > | 18. < T5, H, 3 > |
| 9. < START CKPT ???? > | 19. < START CKPT ???? > |
| 10. < START T4 > | 20. < COMMIT T5 > |

1.
What are the
correct values of
the two
<START CKPT ????>
records?

First START CKPT:
< **START CKPT (T2, T3)** >

Second START CKPT:
< **START CKPT (T4, T5)** >

Problem 2: REDO Logging (Checkpoint)

- | | |
|--------------------------------------|-------------------------------|
| 1. < START T1 > | 11. < T2, E, 5 > |
| 2. < T1, A, 10 > | 12. < COMMIT T2 > |
| 3. < START T2 > | 13. < T3, F, 1 > |
| 4. < T2, B, 5 > | 14. < T4, G, 15 > |
| 5. < T1, C, 7 > | 15. < END CKPT > |
| 6. < START T3 > | 16. < COMMIT T3 > |
| 7. < T3, D, 12 > | 17. < START T5 > |
| 8. < COMMIT T1 > | 18. < T5, H, 3 > |
| 9. < START CKPT T2,T3 > | 19. < START CKPT T4,T5 > |
| 10. < START T4 > | 20. < COMMIT T5 > |

NOTE:

<Commit T3> after
<END CKPT>

**What are we
CKPTing?**

The transactions
that committed
before **<START
CKPT>**

REDO: How far to scan log from the start?

- Identify committed transactions
- Case 1: See <END CKPT> first
 - All committed transactions before <START CKPT (T1.. TK)> are written
 - Consider T1.. Tk, or transactions that started after <START CKPT...>, trace back until earliest <START Ti>

```
<START T1>
<T1, A, 5>
/ <START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT (T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>
```

REDO: How far to scan log from the start?

- **Identify committed transactions**
- **Case 1: See <END CKPT> first**
 - All committed transactions before <START CKPT (T1.. TK)> are written
 - Consider T1.. Tk, or transactions that started after <START CKPT...>, trace back until earliest <START Ti>
- **Case 2: See <START CKPT(T1..TK)> first**
 - Committed transactions before START CKPT might not have been written to disk
 - Find previous <END CKPT> and its matching <START CKPT(S1, ... Sm)>
 - In that previous <START CKPT(S1, ... Sm)>, determine which transactions S1,...Sm has committed – these are ones you will have to redo
 - Find the earliest <START Si> and redo until the end of the log file

Problem 2: REDO Logging

- | | |
|-------------------------|--------------------------|
| 1. < START T1 > | 11. < T2, E, 5 > |
| 2. < T1, A, 10 > | 12. < COMMIT T2 > |
| 3. < START T2 > | 13. < T3, F, 1 > |
| 4. < T2, B, 5 > | 14. < T4, G, 15 > |
| 5. < T1, C, 7 > | 15. < END CKPT > |
| 6. < START T3 > | 16. < COMMIT T3 > |
| 7. < T3, D, 12 > | 17. < START T5 > |
| 8. < COMMIT T1 > | 18. < T5, H, 3 > |
| 9. < START CKPT T2,T3 > | 19. < START CKPT T4,T5 > |
| 10. < START T4 > | 20. < COMMIT T5 > |

2.
What fragment of the log does the recovery manager need to read?

Problem 2: REDO Logging

- | | |
|--------------------------------|---------------------------------|
| 1. < START T1 > | 11. < T2, E, 5 > |
| 2. < T1, A, 10 > | 12. < COMMIT T2 > |
| 3. < START T2 > | 13. < T3, F, 1 > |
| 4. < T2, B, 5 > | 14. < T4, G, 15 > |
| 5. < T1, C, 7 > | 15. < END CKPT > |
| 6. < START T3 > | 16. < COMMIT T3 > |
| 7. < T3, D, 12 > | 17. < START T5 > |
| 8. < COMMIT T1 > | 18. < T5, H, 3 > |
| 9. < START CKPT T2,T3 > | 19. < START CKPT T4,T5 > |
| 10. < START T4 > | 20. < COMMIT T5 > |

2.

What fragment of the log does the recovery manager need to read?

- We know there was a commit for T5.
- In the previous START CKPT, T2 and T3 were the two active transactions. Both transactions committed and must thus be redone.
- T2 was the earliest one

Problem 2: REDO Logging

- | | |
|-------------------------|--------------------------|
| 1. < START T1 > | 11. < T2, E, 5 > |
| 2. < T1, A, 10 > | 12. < COMMIT T2 > |
| 3. < START T2 > | 13. < T3, F, 1 > |
| 4. < T2, B, 5 > | 14. < T4, G, 15 > |
| 5. < T1, C, 7 > | 15. < END CKPT > |
| 6. < START T3 > | 16. < COMMIT T3 > |
| 7. < T3, D, 12 > | 17. < START T5 > |
| 8. < COMMIT T1 > | 18. < T5, H, 3 > |
| 9. < START CKPT T2,T3 > | 19. < START CKPT T4,T5 > |
| 10. < START T4 > | 20. < COMMIT T5 > |

3. Which elements are recovered by the redo recovery manager? compute their values after recovery.

Problem 2: REDO Logging

- | | |
|-------------------------|--------------------------|
| 1. < START T1 > | 11. < T2, E, 5 > |
| 2. < T1, A, 10 > | 12. < COMMIT T2 > |
| 3. < START T2 > | 13. < T3, F, 1 > |
| 4. < T2, B, 5 > | 14. < T4, G, 15 > |
| 5. < T1, C, 7 > | 15. < END CKPT > |
| 6. < START T3 > | 16. < COMMIT T3 > |
| 7. < T3, D, 12 > | 17. < START T5 > |
| 8. < COMMIT T1 > | 18. < T5, H, 3 > |
| 9. < START CKPT T2,T3 > | 19. < START CKPT T4,T5 > |
| 10. < START T4 > | 20. < COMMIT T5 > |

3.
Which elements are recovered by the redo recovery manager? compute their values after recovery.

All changes by T2, T3, T5 (committed)

B=5

D=12

E=5

F=1

H=3

Comparison Undo/Redo

▪ Undo logging:

Steal/Force

- 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

No-Steal/No-Force

- 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)

Steal/No-Force

ARIES Data Structures

▪ Active Transactions Table

- Lists all active TXN's
- For each TXN: **lastLSN** = its most recent update LSN

▪ Dirty Page Table

- Lists all dirty pages
- For each dirty page: **recoveryLSN** (**recLSN**) = first LSN that caused page to become dirty

▪ Write Ahead Log

- LSN, **prevLSN** = previous LSN for same txn

ARIES Data Structures

Dirty page table

pageID	recLSN

LSN

101

Log

prevLSN	tID	pID	Log entry	Type	undoNextLSN

Transaction table

transID	lastLSN	status

Buffer Pool

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Review: ARIES Data Structures (UNDO/REDO Logging)

Example.

1. T_{1000} changes the value of **A** from “abc” to “def” on page P500
2. T_{2000} changes the value of **B** from “hij” to “klm” on page P600
3. T_{2000} changes the value of **D** from “mnp” to “qrs” on page P500
4. T_{1000} changes the value of **C** from “tuv” to “wxy” on page P505
5. T_{2000} commits and the end log record is written
6. T_{1000} changes the value of **E** from “pq” to “rs” on page P700
7. **P600** is flushed to disk
8. **Crash!!**

ARIES Data Structures

Dirty page table

pageID	recLSN

LSN

101

Log

prevLSN	tID	pID	Log entry	Type	undoNextLSN

Transaction table

transID	lastLSN	status

Buffer Pool

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

First operation:

1. T_{1000} changes the value of **A** from “abc” to “def” on **page P500?**

Dirty page table

pageID	recLSN

LSN
101

Log

prevLSN	tID	pID	Log entry	Type	undoNextLSN

Transaction table

transID	lastLSN	status

Buffer Pool

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Changes

1. T_{1000} changes the value of **A** from “abc” to “def” on **page P500**

Dirty page table

pageID	recLSN
P500	101

LSN
101

Log

prevLSN	tID	pID	Log entry	Type	undoNextLSN
-	T1000	P500	Write A “abc” -> “def”	Update	-

Transaction table

transID	lastLSN	status
T_{1000}	101	Running

Buffer Pool

P500 PageLSN= 101 A = def D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Next:

2. T_{2000} changes the value of **B** from “hij” to “klm” on **page P600** ?

Dirty page table

pageID	recLSN
P500	101

LSN
101

Log

prevLSN	tID	pID	Log entry	Type	undoNextLSN
-	T1000	P500	Write A “abc” -> “def”	Update	-

Transaction table

transID	lastLSN	status
T_{1000}	101	Running

Buffer Pool

P500 PageLSN= 101 A = def D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Changes:

2. T_{2000} changes the value of **B** from “hij” to “klm” on **page P600** ?

Dirty page table

pageID	recLSN
P500	101
P600	102

LSN

101

102

Log

prevLSN	tID	pID	Log entry	Type	undoNextLSN
-	T1000	P500	Write A “abc” -> “def”	Update	-
-	T_{2000}	P600	Write B “hij” -> “klm”	Update	

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	102	Running

Buffer Pool

P500 PageLSN= 101 A = def D = mnp	P600 PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Next:

3. T_{2000} changes the value of **D** from “mnp” to “qrs” on **page P500**?

Dirty page table

pageID	recLSN
P500	101
P600	102

LSN

101

102

Log

prevLSN	tID	pID	Log entry	Type	undoNextLSN
-	T1000	P500	Write A “abc” -> “def”	Update	-
-	T_{2000}	P600	Write B “hij” -> “klm”	Update	

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	102	Running

Buffer Pool

P500 PageLSN= 101 A = def D = mnp	P600 PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Changes:

3. T_{2000} changes the value of **D** from “mnp” to “qrs” on page **P500**

Dirty page table

pageID	recLSN
P500	101
P600	102

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	103	Running

Log

LSN

101

102

103

prevLSN	tID	plD	Log entry	Type	undoNextLSN
-	T_{1000}	P500	Write A “abc” -> “def”	Update	-
-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Next:

4. T_{1000} changes the value of **C** from “tuv” to “wxy” on page P505?

Dirty page table

pageID	recLSN
P500	101
P600	102

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	103	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T_{1000}	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
103	102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Changes:

4. T₁₀₀₀ changes the value of C from “tuv” to “wxy” on page P505?

Dirty page table

pageID	recLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T₁₀₀₀	104	Running
T ₂₀₀₀	103	Running

LSN

101

102

103

104

Log

prevLSN	tID	pID	Log entry	Type	undoNextLSN
-	T1000	P500	Write A “abc” -> “def”	Update	-
-	T ₂₀₀₀	P600	Write B “hij” -> “klm”	Update	-
102	T ₂₀₀₀	P500	Write D “mnp” -> “qrs”	Update	-
101	T₁₀₀₀	P505	Write C “tuv” -> “wxy”	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= 104 C = wxy	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Next:

5. T_{2000} commits and the end log record is written

Dirty page table

pageID	recLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T_{1000}	104	Running
T_{2000}	103	Running

LSN

101

102

103

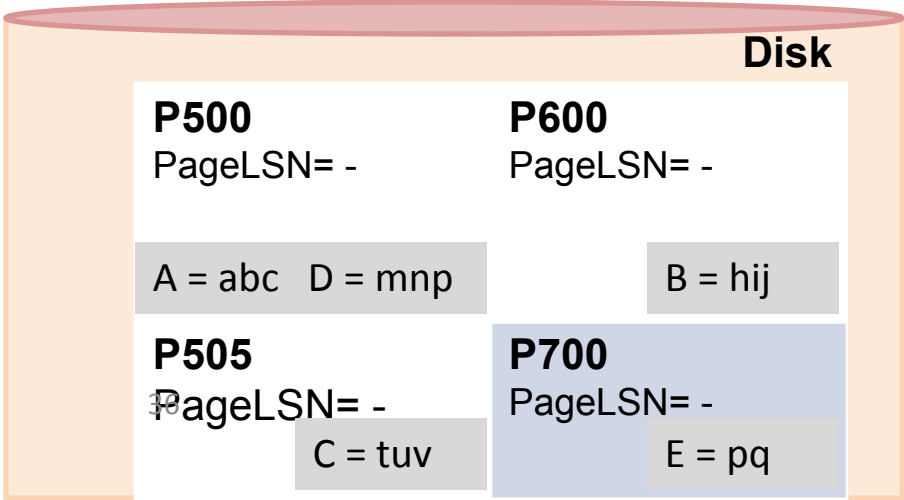
104

Log

prevLSN	tID	pID	Log entry	Type	undoNextLSN
-	T_{1000}	P500	Write A "abc" -> "def"	Update	-
-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= 104 C = wxy	P700 PageLSN= - E = pq



Changes:

5. T_{2000} commits and the end log record is written --- step 1

Log

Dirty page table

pageID	recLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T_{1000}	104	Running
T_{2000}	103	Committed

LSN

101

102

103

104

105

106

prevLSN	tID	pID	Log entry	Type	undoNextLSN
-	T_{1000}	P500	Write A "abc" -> "def"	Update	-
-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
103	T_{2000}			Commit	
105	T_{2000}			End	

Buffer Pool

P500

PageLSN= 103

P600

PageLSN= 102

A = def D = qrs

B = klm

P505

PageLSN= 104

C = wxy

P700

PageLSN= -

E = pq

Disk

P500

PageLSN= -

A = abc D = mnp

P600

PageLSN= -

B = hij

P505

PageLSN= -

C = tuv

P700

PageLSN= -

E = pq

Changes:

5. **T₂₀₀₀** commits and the end log record is written --- step 2

Log

Dirty page table

pageID	recLSN
P500	101
P600	102
P505	104

T2000 removed from transaction table

Transaction table

transID	lastLSN	status
T ₁₀₀₀	104	Running
T₂₀₀₀	103	Committed

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T ₁₀₀₀	P500	Write A "abc" -> "def"	Update	-
102	101	T ₁₀₀₀	P600	Write B "hij" -> "klm"	Update	-
103	102	T ₂₀₀₀	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T ₁₀₀₀	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T ₂₀₀₀			Commit	
106	105	T ₂₀₀₀			End	

Buffer Pool

P500

PageLSN= 103

P600

PageLSN= 102

A = def D = qrs

B = klm

P505

PageLSN= 104

C = wxy

P700

PageLSN= -

E = pq

Disk

P500

PageLSN= -

A = abc D = mnp

P600

PageLSN= -

B = hij

P505

PageLSN= -

C = tuv

P700

PageLSN= -

E = pq

Changes:

5. T_{2000} commits and the end log record is written --- step 2

Dirty page table

pageID	recLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T_{1000}	104	Running
T_{2000}	103	Committed

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T_{1000}	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Buffer Pool

P500

PageLSN= 103

A = def D = qrs

P505

PageLSN= 104

C = wxy

P600

PageLSN= 102

B = klm

P700

PageLSN= -

E = pq

Disk

Log written to disk

Note: no force = not the dirty
pages changed by T_{2000} !

A = abc D = mnp

B = hij

P505

PageLSN= -

C = tuv

P700

PageLSN= -

E = pq

Whenever a transaction commits,
log is flushed to the disk == the log-tail is written to disk

NOTE:

1. The “Commit” record is required to be flushed (i.e. all logs up to and including that commit record)
2. The “End” record is not required to be flushed, in this case we are only assuming that it has been flushed as well (so that we have a good example while doing recovery 😊)

Next:

6. T_{1000} changes the value of **E** from “pq” to “rs” on **page P700**

Dirty page table

pageID	recLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T_{1000}	104	Running

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T_{1000}	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
103	102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-
104	101	T_{1000}	P505	Write C “tuv” -> “wxy”	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Buffer Pool

P500

PageLSN= 103

A = def D = qrs

P505

PageLSN= 104

C = wxy

P600

PageLSN= 102

B = klm

P700

PageLSN= -

E = pq

Disk

P500

PageLSN= -

A = abc D = mnp

P505

PageLSN= -

C = tuv

P600

PageLSN= -

B = hij

P700

PageLSN= -

E = pq

Changes:

6. T_{1000} changes the value of **E** from “pq” to “rs” on page P700
Log

Dirty page table

pageID	recLSN
P500	101
P600	102
P505	104
P700	107

Transaction table

transID	lastLSN	status
T_{1000}	107	Running

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T_{1000}	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
103	102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-
104	101	T_{1000}	P505	Write C “tuv” -> “wxy”	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107	104	T_{1000}	P700	Write E “pq” -> “rs”	Update	-

Buffer Pool

P500

PageLSN= 103

P600

PageLSN= 102

A = def D = qrs

B = klm

P505

PageLSN= 104

C = wxy

P700

PageLSN= 107

E = rs

Disk

P500

PageLSN= -

P600

PageLSN= -

A = abc D = mnp

B = hij

P505

PageLSN= -

C = tuv

P700

PageLSN= -

E = pq

Next:

7. Page P600 is flushed to disk

Log

Dirty page table

pageID	recLSN
P500	101
P600	102
P505	104
P700	107

Transaction table

transID	lastLSN	status
T ₁₀₀₀	107	Running

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T ₁₀₀₀	P500	Write A "abc" -> "def"	Update	-
102	-	T ₂₀₀₀	P600	Write B "hij" -> "klm"	Update	-
103	102	T ₂₀₀₀	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T ₁₀₀₀	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T ₂₀₀₀			Commit	
106	105	T ₂₀₀₀			End	
107	104	T ₁₀₀₀	P700	Write E "pq" -> "rs"	Update	-

Buffer Pool

P500

PageLSN= 103

P600

PageLSN= 102

A = def D = qrs

B = klm

P505

PageLSN= 104

C = wxy

P700

PageLSN= 107

E = rs

Disk

P500

PageLSN= -

P600

PageLSN= -

A = abc D = mnp

B = hij

P505

PageLSN= -

C = tuv

P700

PageLSN= -

E = pq

Next:

7. Page P600 is flushed to disk – Step 1

Dirty page table

pageID	recLSN
P500	101
P600	102
P505	104
P700	107

Transaction table

transID	lastLSN	status
T ₁₀₀₀	107	Running

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T ₁₀₀₀	P500	Write A “abc” -> “def”	Update	-
102	-	T ₂₀₀₀	P600	Write B “hij” -> “klm”	Update	-
103	102	T ₂₀₀₀	P500	Write D “mnp” -> “qrs”	Update	-
104	101	T ₁₀₀₀	P505	Write C “tuv” -> “wxy”	Update	-
105	103	T ₂₀₀₀			Commit	
106	105	T ₂₀₀₀			End	
107	104	T ₁₀₀₀	P700	Write E “pq” -> “rs”	Update	-

Buffer Pool

P500

PageLSN= 103

A = def D = qrs

P505

PageLSN= 104

C = wxy

P600

PageLSN= 102

B = klm

P700

PageLSN= 107

E = rs

Disk

P500

PageLSN= -

A = abc D = mnp

P505

PageLSN= -

C = tuv

P600

PageLSN= 102

B = klm

P700

PageLSN= -

E = pq

Next:

7. Page P600 is flushed to disk – Step 2

Dirty page table

pageID	recLSN
P500	101
P505	104
P700	107

Transaction table

transID	lastLSN	status
T ₁₀₀₀	107	Running

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T ₁₀₀₀	P500	Write A “abc” -> “def”	Update	-
102	-	T ₂₀₀₀	P600	Write B “hij” -> “klm”	Update	-
103	102	T ₂₀₀₀	P500	Write D “mnp” -> “qrs”	Update	-
104	101	T ₁₀₀₀	P505	Write C “tuv” -> “wxy”	Update	-
105	103	T ₂₀₀₀			Commit	
106	105	T ₂₀₀₀			End	
107	104	T ₁₀₀₀	P700	Write E “pq” -> “rs”	Update	-

Buffer Pool

P500

PageLSN= 103

A = def D = qrs

P505

PageLSN= 104

C = wxy

P600

PageLSN= 102

B = klm

P700

PageLSN= 107

E = rs

Disk

P500

PageLSN= -

A = abc D = mnp

P505

PageLSN= -

C = tuv

P600

PageLSN= 102

B = klm

P700

PageLSN= -

E = pq

8. CRASH!!



8. Crash!! ---- These are gone from memory

Dirty page table

P700

Transaction table

transID	lastLSN	status

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T ₂₀₀₀	P600	Write B "hij" -> "klm"	Update	-
103	102	T ₂₀₀₀	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T ₁₀₀₀	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T ₂₀₀₀			Commit	
106	105	T ₂₀₀₀			End	
107						

Buffer Pool

PageLSN= 103

A = def

P505

PageLSN=

C = wxy

N= 107

E = rs

Disk

P500

PageLSN= -

A = abc D = mnp

P505

PageLSN= -

C = tuv

P600

PageLSN= 102

B = klm

P700

PageLSN= -

E = pq