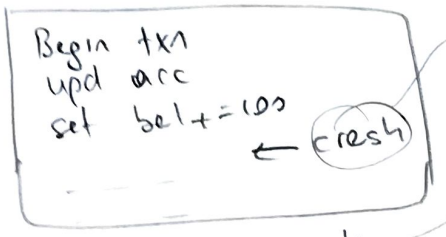
week 8

TRANSACTIONS : RECOVERY

Concurrency = A C I D → isolation

Recovery = A C I D → durability
→ atomicity



→ rollback, atomicity

→ durability

← crash

→ they should be durable

→ they should be aborted

- System failures → dB recovery
- When a crash happens we dont know which part of the txn is executed and which didn't.

Java —Read&Write requests→ DBMS —input/output→ OS —fetches data pages→ Buffer of DBMS (like memory)

- If the buffer is full → replacement policy (oldest-one etc.)
- DBMS implements buffer
- Buffer pool knows which frame is which page id.

Problems during crash
→ committed data is not written to disk
→ uncommitted data in the disk.

R(x,t) → x to local
W(x,t) → local to x
I(x) → read from disk
O(x) → write to disk

ex:

R(A,t), t*2, W(A,t)
R(B,t), t*2, W(B,t)

Action	t	Buffer: Mem A	Buffer: Mem B	Disk: Disk A	Disk: Disk B
Input(A)		8		8	8
R(A,t)	8	8		8	8
t*2	16	8		8	8
W(A,t)	16	16		8	8
I(B)	16	16	8	8	8
R(B,t)	8	16	8	8	8
t*2	16	16	8	8	8
W(B,t)	16	16	16	8	8
3 O(A)	16	16	16	16	8
1 O(B)	16	16	16	16	16
2 Commit					

1 A is written to disk but B is not (All or none). A should be back to 8.

2 Commit is not sent to db. But it is written to the disk. How can dbms know that the next action is commit?

3 We haven't sent them to disk. Memory is gone, no problem.

*Output can also after commit

Commit					
O(A)	16	16	16	16	8
O(B)	16	16	16	16	16

↳ To release the locks so that concurrency continues.

DBMS responsability for committed transactions to REDO all steps.
↳ make sure it is there (durability)

Undoing → atomicity Redoing → durability

Force / No Force

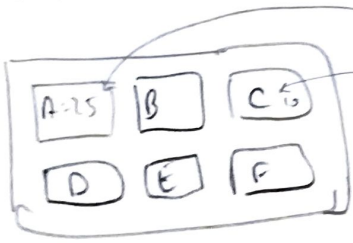
When dbms receives <u>commit</u>, should it flush pages to disk right now?

Force → yes, costly in timewise

No force → dont rush.

Steal / No Steal

Least Recently used frame is sent to disk (because buffer is full) to retrieve a new page from disk.



A = 25

C = C * 2

G = G * 2 → one of the pages must be sent to disk
- → A is the oldest one, sent it.
- → Uncommitted action to disk (might dirty-read)

Steal → sent a frame to disk. We dont want a txn to wait just because the buffer is full.

No steal → Txn will be delayed, bad response time.

Force / No steal → easy to implement, safe
- → terrible response time

We want <u>no force / steal</u>.

We need log (Write Ahead Log, WAL). Whenever you are gonna change the disk, write into Log. Log file is in the memory, periodically flushed to disk.

WAL

- append only
- every action of every txn
- UNDO, REDO, UNDO-REDO
- use log to recover.

	No steal	steal
Force	x	undo log
No Force	redo log	undo-redo

Undo-Redo

<Start T>

<commit T>

<abort T>

<T, X, u, v> → T updated el. X, old val = u
new val = v

	<Start T>
Input(A)	
R(A,t)	
t*2	<T, A, 8, 16>
Write(A,t)	
	<T, B, 8, 16> *
W(B,t)	
out(A)	<commit T>
out(B)	
Commit	* 2

*Crash → a. is written to disk but B is not.
Undo. by B=8, A=8, using log. → Undo

Before accepting a new txn, A and B should be 8.

*2 Recovery must make sure that A=B=16, Redo

*We are free to OUTPUT before or after <commit T>

RULE*

If T modifies X, log <T, X, u, v> must be written to disk before output(x).

i.e. → when dbms requested output(x), <T, X, u, v> must be sent to the disk before writing X to the disk.

1) Scan log, top to down ↓, redo all committed txn.
2) Undo log bottom to up ↑, undo all uncommitted txn

Q1 Which updates are undone

Log

<T6, X6, --->

Start T5
Start T4

T1, X1
T5, X5
T4, X4
Commit T5
T3, X3
T2 X2

crash

1) T6 running
2) T5 running
3) T4 run
4) T1 run
5) T5 run
6) T4 run
7) T5 commit
8) T3 run
9) T2 run

- T5 should be redo
- others undo.

Q2 How far do we need to go back in the log?

- Use checkpoints.

Q3 What happens when 2nd crash during recovery?

- Write what you do in recovery

Checkpoint

- Checkpoint periodically
- Stop accepting new txn., wait until all current txn finish.
- Flush log to disk.
- write <CKPT>
- resume txn's.

Log

<ckpt>

} all completed

} others, redo or undo them

Problem → during checkpoint, db freezes

we want non-freezing (non-quiscent) checkpoint

ARIES

- redo, undo info for every update in Log.
- < xid, pageid, offset, length, old, new >
  - xid → txn id
  - offset → in the page
  - length → length of change

<start T>
< T, x, u, v>
<commit T> or <abort T> then <End>
<clr> → for undoing

<u>Some rule</u> → Before writing X to disk, <T, x, u, v> log to disk.

LSN Log sequence number
↳ Log entries belonging same txn are linked.

- each page has pageLSN → which log (?) has the last update (in the disk)

<u>Data Structures</u>

Active txn table
- active txn's
- for each txn, Last LSN → most recent update LSN and status (run, com, abo)

Dirty Page table
- all dirty pages
- each dirty page, recovery LSN → <u>first</u> LSN that caused page to dirty

WAL
- LSN, pre-LSN ≠ prev LSN for same txn.

ex

1) Dirty page

Log

100

active txn

buffer

P1	P2	

initial state

2) $W_{T100}(P7)$

D.P.

pageId	recLSN
P7	101

Log

sln	prevLsn	xid	pid	Logentry
101	–	T100	P7	

active txn

xid	lastlsn
T100	101

Buffer

pageLSN=101 P7	

3) $W_{T200}(P5)$

D.P

p.id	recLsn
P7	101
P5	102

Log

sln	p.sln	xid	pid	
101	–	T100	P7	
102	–	T200	P5	

active txn

xid	lastlsn
T100	101
T200	102

Buffer

pageLSN=101 P7	pageLsn=102 P5

4) $W_{T200}(P6)$

D.P

pid	recLsn
P7	101
P5	102
P6	103

Log

sln	prevsln	xid	pid	
101	–	T100	P7	
102	–	T200	P5	
103	102	T200	P6	

active txn

xid	Lastlsn
T100	101
T200	103

Buffer

	pageLsn: 103 P6
P7	102 P5

5) $W_{T100}(P5)$

did not change

d.p.

pid	recLsn
P7	101
P5	102
P6	103

Log

sln	prevsln	xid	pid	
101	—	T100	P7	
102	—	T200	P5	
103	102	T200	P6	
104	101	T100	P5	

T100

active txn

tid	lastlsn
T100	104
~~T200~~	103

Buffer

104		
P5		

- Buffer manager wants to OUTPUT(P) → ex: bunu diske yazmak istiyor. log'da 104'e kadar olan disk'e yaz.
  - ↳ Flush log up to pageLSN.
  - ↳ remove P from dirty pages table, buffer.

- Buffer wants Input(P)
  - ↳ create entry in Dirty pages, recLSN=null

- a new txn starts
  - ↳ write <Start T> to log
  - ↳ in active txn, new entry, lastlsn=null

- a txn commits → page may not be sent to the disk yet. (log must sent to disk!)
  - ↳ write <Commit T> to log (in memory)
  - ↳ flush log up to this entry
  - ↳ write <end>

Fuzzy Checkpointing

- write into log
  - → entire active txn table
  - → " dirty page table
- recovery starts from last checkpoint
- background process periodically flushes dirty pages to disk.

Checkpointing in Aries

begin ckpt

end ckpt
- → current txn, dirt page table
- → fuzzy checkpoint
- → other txn's continue.

each data page has last pageLsn.

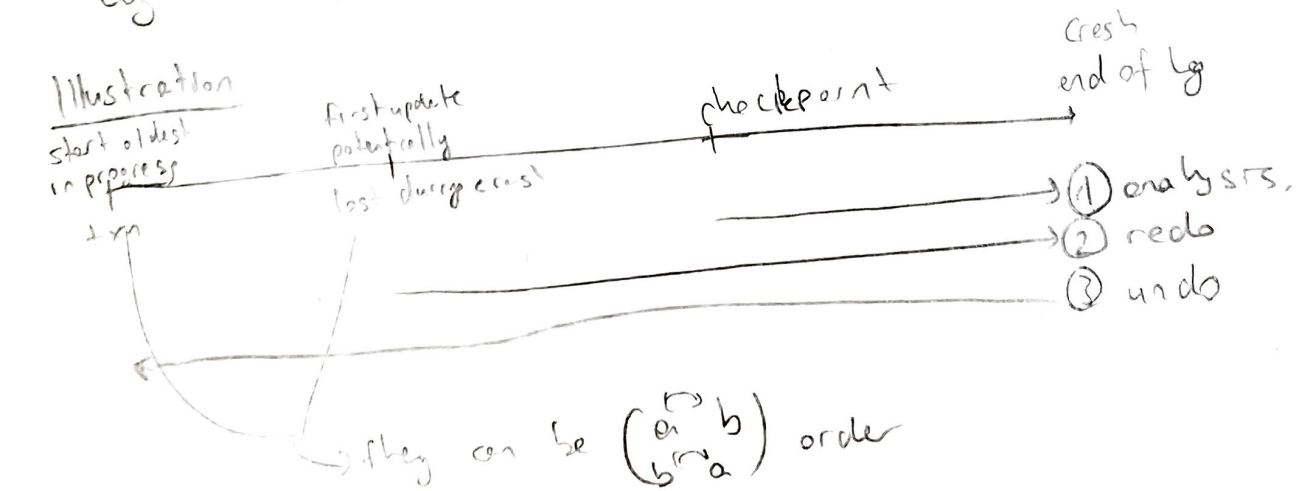
ARIES Recovery

1) Analysis → from checkpoint, build dirty page / active txn table.
2) Redo pass → redo all operations, even txn that will not commit.

   redo might be before checkpoint!

3) Undo pass
   Remove effects of uncommitted txn.

   Log the changes in case of another crash.



1) Analysis Phase

- Figure out starting point of redo (smallest LSN)

firstLSN — chpoint — crash

? → By going, find min recLSN

Build active txn, dirty page tables.

where does redo start?

2) Redo Phase

- Replay history, starting from firstLSN.
- Doing redo, we dont need to write in to log.
  ↳ Bir daha crash olursa tekrar redo.

For each LSN <T, P, u, v>

redo action P=u, write P.

- If P is not in Dirty Page, no update
- if recLSN > LSN, no update
- read page from disk
  ↳ if pageLSN ≥ LSN, no update

} ?

ow. update

3) Undo Phase

- Undo only the loser txn.
- Selective undo.
- move backwards
- undo actions → CLR in the log (compensation log records)
- CLR are redone, but never undone ?

- System crash during undo
  - ↳ not undo again.
  - ↳ each CLR is a REDO record, redo the undo.

ex:

	LSN	Log
1	00	begin checkpoint
	05	end checkpoint
2	10	upd T1 → P5
3	20	upd T2, P3
4	30	T1 abort
5	40	CLR undo T1 SLN 10
6	45	T1 end
7	50	upd T3 → P1
8	60	upd T2 → P5

Crash

1) Analysis

1 dp table

active txn

2 dp

pid	recLSN
P5	10

active txn

xid	LSN	status
T1	10	run

3 dp

pid	recLSN
P5	10
P3	20

active txn

xid	LSN	status
T1	10	run
T2	20	run

4 dp

pid	recLSN
P5	10
P3	20

active txn

xid	LSN	status
T1	30	abort
T2	20	run

5 dp

pid	recLSN
P5	10
P3	20

active txn

xid	LSN	status
T1	40	abort
T2	20	run

6 dp

pid	recLSN
P5	10
P3	20

active txn

xid	LSN	status
T2	20	run

T1 is removed

7 dp

pid	recLSN
P5	10
P3	20
P1	50

active txn

xid	LSN	stat
T2	20	run
T3	50	run

8 dp

pid	recLSN
P5	10
P3	20
P1	50

still 10

active txn

xid	LSN	stat
T2	60	run
T3	50	run

final

2) Redo step → By looking at the log

dp

pid	recLsn
P5	10
P3	20
P1	50

active txn

xid	Lsn	stat
T2	60	run
T3	50	run

① start from 10.
Is P5 in dirty page ⇒ no. Bring P5 to buffer, update in memory.

Buffer

10	P5*

② Bring P3

Buffer

10	20
P5*	P3*

④-⑤ clr undo T1

40	20
P5*	P3*

⑦ T3 writes P1

40	70	50
P5*	P3*	P1*

⑧ T_2 writes P5

50	20	50
P5*	P3*	P1*

end of redo

3) Undo step

active txn {T2, T3}, they must be undone (backwards)

↳ undo {60, 50}

① log

60	upd T2 → P5
70	CLR undo T2 LSN 60

① T2 writes P5

Buffer

70	20	50
P5	P3*	P1*

② In the log, there was link from 60 to 20. T_2'nin bir onceki 20 imis.

undo {~~60~~, 50, 20}

③ Undo 50, T3 writes P1

Log

-	
70	-
80	CLR Undo T3 LSN 50

Buffer

70	20	80
P5	P3*	P1

④ 50'den geriye link yok.

Undo {~~50~~, 20}

Undo 20, T2 writes P3

Log

70	
80	
90	CLR undo T2 LSN 20

Buffer

~~70~~	90	80
P5	P3	P1

- No link before 20. undo list is empty, done

(in this example we did not go back to checkpoint (gidebilirdik).

* → degismisti, undo yapinca * kalkiyor.

- Su ana kadar memorydeydi.

↳ Flush to disk (Buffer + Log)