CSE232: Database System Principles

Failure Recovery

1

Integrity or correctness of data

- Would like data to be "accurate" or "correct" at all times

EMP

Name	Age
White	52
Green	3421
Gray	1

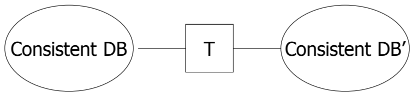
2

Integrity or consistency constraints

- Predicates DB data must satisfy
  - e.g., x is key of relation R
  - $x \rightarrow y$ holds in R
  - Domain(x) = {Red, Blue, Green}
  - no employee should make more than twice the average salary
- Application business logic implies pre-post transaction constraints on DB
  - Eg, value of Joe's checking account after the deposit of $X is the prior value + X

3

Transaction: collection of actions
that preserve DB consistency



4

Big working assumption:

If T starts with consistent state +
T executes until completion
& in isolation
⇒ T leaves consistent state

5

How we will break the assumptions on
T's execution and lead to **in**correctness:

If T starts with consistent state +
T executes until ~~completion~~ **FAILURES**
& in ~~isolation~~ **CONCURRENT EXECUTION WITH DATA SHARING**
⇒ T leaves consistent state

6

How can we prevent/fix violations? Preview of the next episodes:

- Failure Recovery: fixing violations due to failures only
- Concurrency Control: fixing violations due to concurrency & data sharing only
- finally a mix of the two: fixing violations that are stem from interaction of failures with sharing

7

We will not consider in CSE232:

- How to write correct transactions
  - A buggy transaction can violate constraints even if it runs to completion, in isolation
- How to write correct DBMS
  - A correct transaction running to completion & in isolation can violate constraints if the DB's query processor has bugs

8

Failures & Recovery

- First order of business:
  Failure Model

9

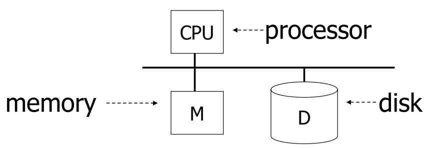
Events — Desired

Undesired — Expected

Unexpected

10

Our failure model



11

Desired events: see product manuals….

Undesired expected events:

System crash

- memory lost
- cpu halts, resets

that's it!!

Undesired Unexpected: Everything else!

12

Undesired Unexpected: Everything else!

Examples:
- Disk data is lost
- Memory lost without CPU halt
- Skynet's CPU decides to wipe out its programmers….

13

Is this model reasonable?

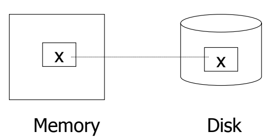
Approach: Add low level checks + redundancy to increase probability model holds

E.g., Replicate disk storage (stable store)
Memory parity
CPU checks

14

Second order of business:

Storage hierarchy



15

Operations:

- Input (x): block with x → memory
- Output (x): block with x → disk

- Read (x,t): do input(x) if necessary
  t ← value of x in block
- Write (x,t): do input(x) if necessary
  value of x in block ← t

16

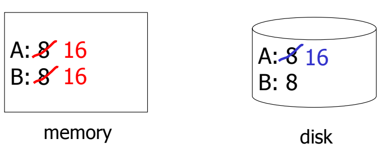
Key problem Unfinished transaction

Example Constraint: A=B

T1: A ← A × 2
  B ← B × 2

17

T1: Read (A,t); t ← t×2
 Write (A,t);
 Read (B,t); t ← t×2
 Write (B,t);
 Output (A);
 Output (B); failure!



18

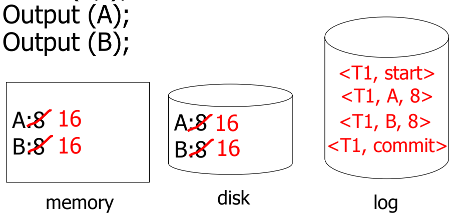
- Need <u>atomicity:</u> execute all actions of a transaction or none at all

<u>One solution:</u> undo logging (immediate modification)

due to: Hansel and Gretel, 782 AD

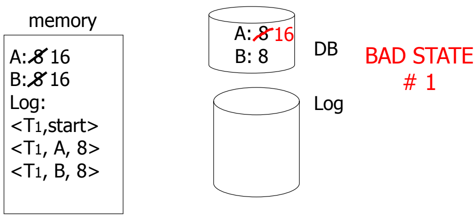
Undo logging (Immediate modification)

T1: Read (A,t); $t \leftarrow t \times 2$ A=B
Write (A,t);
Read (B,t); $t \leftarrow t \times 2$
Write (B,t);
Output (A);
Output (B);



One "complication"

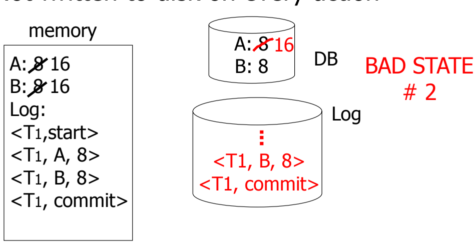
- Log is first written in memory
- Not written to disk on every action



22

Two "complications"

- Log is first written in memory
- Not written to disk on every action



23

Undo logging rules

(1) For every action generate undo log record (containing old value)

(2) Before x is modified on disk, log records pertaining to x must be on disk (write ahead logging: WAL)

(3) Before commit is flushed to log, all writes of transaction must be reflected on disk

24

Recovery rules, Take One:
Undo logging

- For every Ti with <Ti, start> in log:
  - If <Ti,commit> or <Ti,abort> in log, do nothing
  - Else
    - For all <Ti, *X, v*> in log:
      - write *(X, v)*
      - output (*X*)
    - Write <Ti, abort> to log

☒IS THIS CORRECT??

25

Recovery rules: Undo logging

(1) Let S = set of transactions with
<Ti, start> in log, but no
<Ti, commit> (or <Ti, abort>) record in log

(2) For each <Ti, X, v> in log,
in reverse order (latest → earliest) do:
- if Ti ∈ S then
  - write (X, v)
  - output (X)

(3) For each Ti ∈ S do
- write <Ti, abort> to log

26

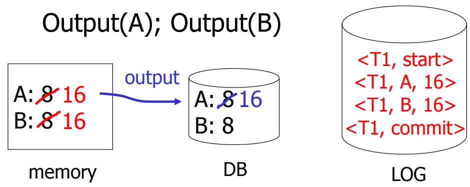
What if failure during recovery?

No problem! ☞ Undo idempotent

27

Redo logging (deferred modification)

T1: Read(A,t); t← t×2; write (A,t);
Read(B,t); t←t×2; write (B,t);
Output(A); Output(B)



Redo logging rules

(1) For every action, generate redo log record (containing new value)

(2) Before X is modified on disk (DB), all log records for transaction that modified X (including commit) must be on disk

(3) Flush log at commit

Recovery rules: Redo logging

- For every Ti with <Ti, commit> in log:
  - For all <Ti, X, v> in log:
    - Write(X, v)
    - Output(X)

☒IS THIS CORRECT??

Recovery rules: Redo logging

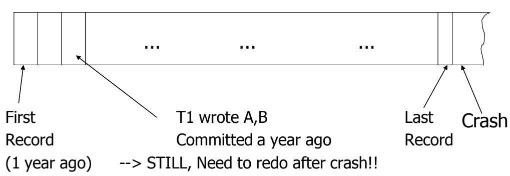
(1) Let S = set of transactions with
<Ti, commit> in log

(2) For each <Ti, X, v> in log, in forward order (earliest → latest) do:

- if Ti $\in$ S then $\begin{cases} \text{Write(X, v)} \\ \text{Output(X)} \leftarrow \text{optional} \end{cases}$

Recovery is very, very SLOW !

Redo log:



Solution: Checkpoint (simple version)

Periodically:

(1) Do not accept new transactions

(2) Wait until all transactions finish

(3) Flush all log records to disk (log)

(4) Flush all buffers to disk (DB) (do not discard buffers)

(5) Write "checkpoint" record on disk (log)

(6) Resume transaction processing

Example: what to do at recovery?

Redo log (disk):

...	<T1,A,16>	...	<T1,commit>	...	Checkpoint	...	<T2,B,17>	...	<T2,commit>	...	<T3,C,21>	Crash

Key drawbacks:

- *Undo logging:*
  cannot bring backup DB copies up to date,
  real writes at end of transaction needed
- *Redo logging:*
  need to keep all modified blocks in memory
  until commit

Solution: undo/redo logging!

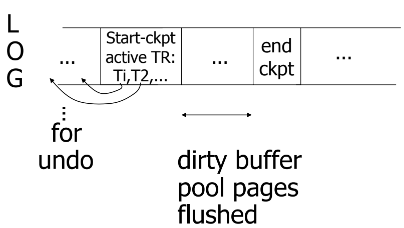
Update $\Rightarrow$ <Ti, Xid, New X val, Old X val>
page X

Rules

- Page X can be flushed before or after Ti commit
- Log record flushed before corresponding updated page (WAL)
- Flush at commit (log only)

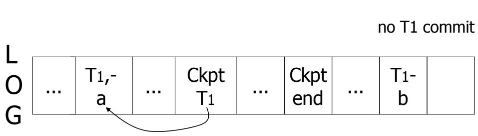
37

Non-quiesce checkpoint



38

Examples what to do at recovery time?



☒ Undo T1 (undo a,b)

39

Example

L O G

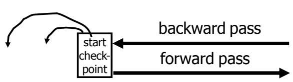
...	T_1 a	...	ckpt-s T_1	...	T_1 b	...	ckpt-end	...	T_1 c	...	T_1 cmt	...

☒ Redo T1: (redo b,c)

40

Recovery process:

- Backwards pass (end of log ➲ latest checkpoint start)
  - construct set S of committed transactions
  - undo actions of transactions not in S
- Undo pending transactions
  - follow undo chains for transactions in (checkpoint active list) - S
- Forward pass (latest checkpoint start ➲ end of log)
  - redo actions of S transactions



41

Real world actions

E.g., dispense cash at ATM

$$Ti = a_1\, a_2 \ldots\ldots a_j \ldots\ldots a_n$$

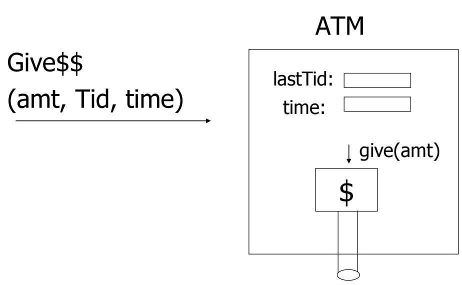
$$\downarrow$$

$

42

Solution

(1) execute real-world actions after commit
(2) try to make idempotent



Summary

- Consistency of data
- One source of problems: failures
  - Logging
  - Redundancy
- Next source of problems:
  Concurrency + Data Sharing