CSE 562
Database Systems

Failure Recovery

Some slides are based or modified from originals by
Database Systems: The Complete Book,
Pearson Prentice Hall 2nd Edition
©2008 Garcia-Molina, Ullman, and Widom

cse@buffalo

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

Integrity or Consistency Constraints

- Predicates data must satisfy
- Examples:
  - x is key of relation R
  - $x \rightarrow y$ holds in R
  - Domain(x) = {Red, Blue, Green}
  - α is valid index for attribute x of R
  - no employee should make more than twice the average salary

Definition

- Consistent state: satisfies all constraints
- Consistent DB: DB in consistent state

Constraints (as we use here) may not capture "full correctness"

Example 1: Transaction constraints

- When salary is updated,
  new salary > old salary
- When account record is deleted,
  balance = 0

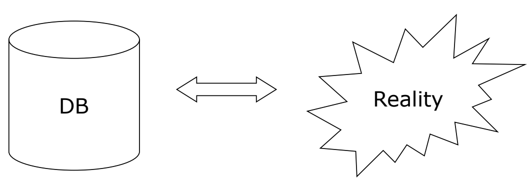
Note: could be "emulated" by simple constraints, e.g.,

account

Acct #	....	balance	deleted?

Constraints (as we use here) may not capture "full correctness"

Example 2: Database should reflect real world



In any case, continue with constraints...

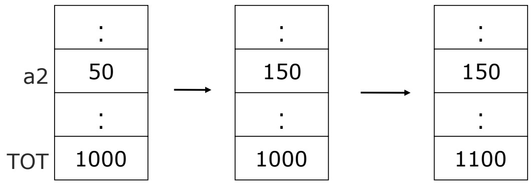
Observation: DB cannot be consistent always!

Example: a1 + a2 +.... an = TOT (constraint)

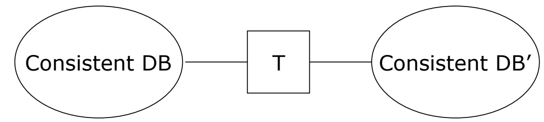
Deposit \$100 in a2: $\begin{cases} a2 \leftarrow a2 + 100 \\ TOT \leftarrow TOT + 100 \end{cases}$

Example: a1 + a2 +.... an = TOT (constraint)

Deposit \$100 in a2: $\begin{cases} a2 \leftarrow a2 + 100 \\ TOT \leftarrow TOT + 100 \end{cases}$



Transaction: collection of actions
that preserve consistency



Big Assumption

If T starts with consistent state +
T executes in isolation
$\Rightarrow$ T leaves consistent state

Correctness (informally)

- If we stop running transactions, DB left consistent
- Each transaction sees a consistent DB

How Can Constraints Be Violated?

- Transaction bug
- DBMS bug
- Hardware failure
  - e.g., disk crash alters balance of account
- Data sharing, e.g.:

  T1: give 10% raise to programmers
  T2: change programmers ⇒ systems analysts

How Can We <u>Prevent/Fix</u> Violations?

- Chapter 17: Due to failures <u>only</u>
- Chapter 18: Due to data sharing <u>only</u>
- Chapter 19: Due to failures and sharing

Will Not Consider

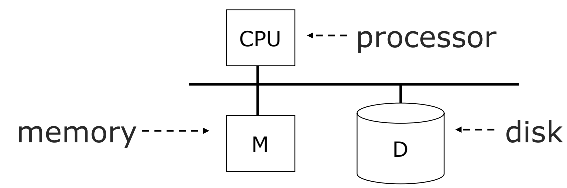
- How to write correct transactions
- How to write correct DBMS
- Constraint checking & repair
  - That is, solutions studied here do not need to know constraints

Chapter 17: Recovery

- First order of business:
  <u>Failure Model</u>

Events — Desired
Undesired — Expected
Unexpected

Our Failure Model



Desired events: see product manuals….

Undesired expected events:

System crash

- memory lost
- cpu halts, resets

that's it!!

Undesired Unexpected: Everything else!

Undesired Unexpected: Everything else!

Examples:

- Disk data is lost
- Memory lost without CPU halt
- CPU implodes wiping out universe…

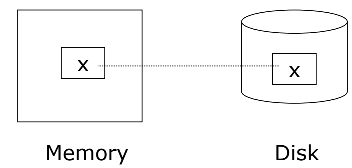
Is This Model Reasonable?

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

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

Second Order of Business:

Storage hierarchy



Operations

- Input (x): block containing x → memory
- Output (x): block containing 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

Key Problem: Unfinished Transaction

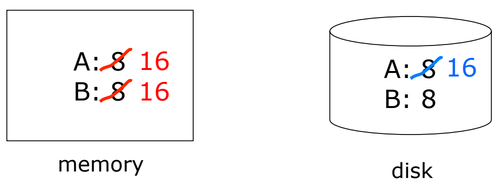
Example Constraint: A=B

T1: $A \leftarrow A \times 2$

$B \leftarrow B \times 2$

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

failure!



- Need atomicity: execute all actions of a transaction or none at all

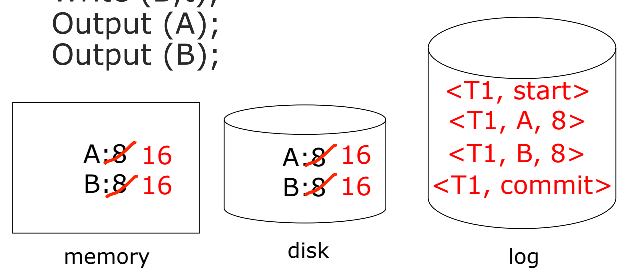
One solution: undo logging

(immediate modification)

due to: Hansel and Gretel, 782 AD

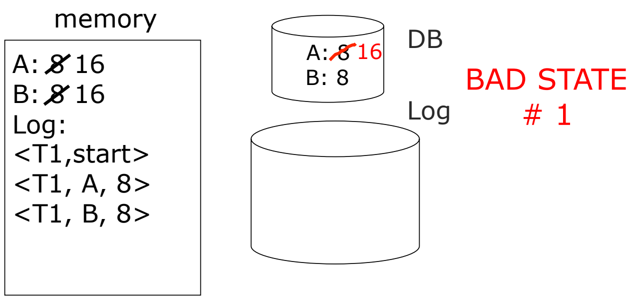
Undo Logging (Immediate modification)

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



One “Complication”

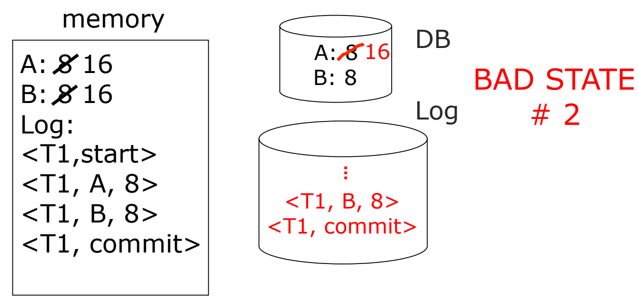
- Log is first written in memory
- Not written to disk on every action



UB CSE 562 29

One “Complication”

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



UB CSE 562 30

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

UB CSE 562 31

Recovery Rules: 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??

UB CSE 562 32

Recovery Rules: Undo Logging

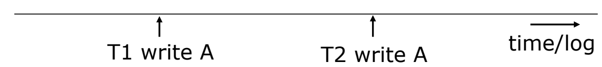
(1) Let S = set of transactions with
<Ti, start> in log, but no
<Ti, commit> (or <Ti, abort>) record in log

(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

Question

- Can writes of <Ti, abort> records be done in any order (in Step 3)?
  - **Example:** T1 and T2 both write A
  - T1 executed before T2
  - T1 and T2 both rolled-back
  - <T1, abort> written but NOT <T2, abort>



What if failure during recovery?

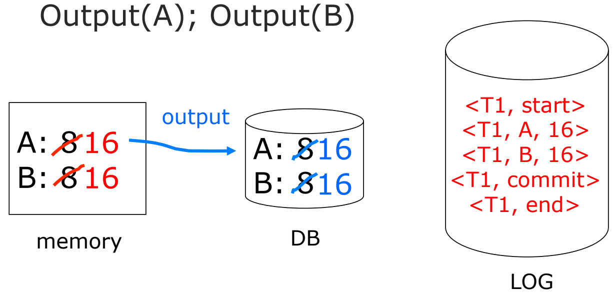
No problem! Undo **idempotent!**

To Discuss:

- Redo logging
- Undo/redo logging, why both?
- Real world actions
- Checkpoints
- Media failures

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

(4) Write END record after DB updates flushed to disk

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> (and no <Ti, end>) in log

(2) For each <Ti, X, v> in log, in forward order (earliest → latest) do:
- If Ti ∈ S then Write(X, v)
  Output(X)

(3) For each Ti ∈ S, write <Ti, end>

Combining <Ti, end> Records

- Want to delay DB flushes for hot objects

Say X is branch balance:
T1: … update X…
T2: … update X…
T3: … update X…
T4: … update X…

Actions:
write X
~~output X~~
write X
~~output X~~
write X
~~output X~~
write X
output X
combined <end> **(checkpoint)**

Solution: Checkpoint

- no <ti, end> actions
- simple checkpoint

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
- *Redo logging:* need to keep all modified blocks in memory until commit

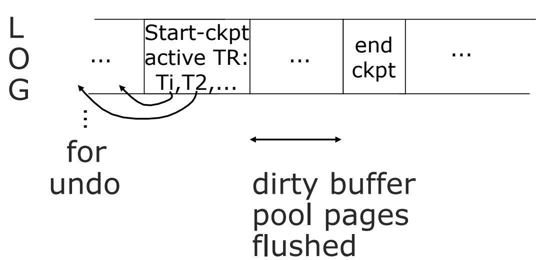
Solution: Undo/Redo Logging!

Update ⇒ <Ti, Xid, New X val, Old X val>
page X

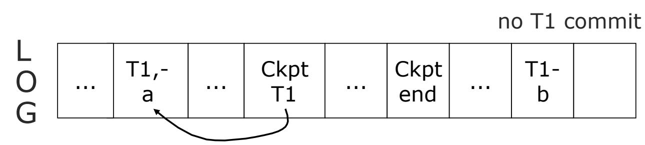
Rules

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

Non-Quiesce Checkpoint



Example: What To Do At Recovery Time?



Undo T1 (undo a,b)

Example

L
O
G

…	T1 a	…	ckpt-s T1	…	T1 b	…	ckpt-end	…	T1 c	…	T1 cmt	…

Redo T1 (redo b,c)

Recover From Valid Checkpoint

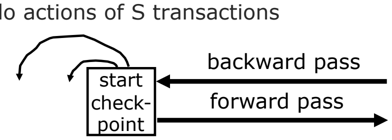
L
O
G

…	ckpt start	…	ckpt end	…	T1 b	…	ckpt-start	…	T1 c	…		

start
of latest
valid
checkpoint

Recovery Process

- **Backwards pass**
  (end of log ➜ latest valid 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



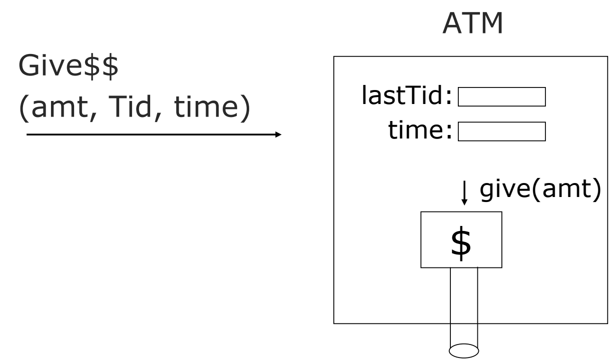
Real World Actions

E.g., dispense cash at ATM
Ti = a1 a2 … … aj … … an

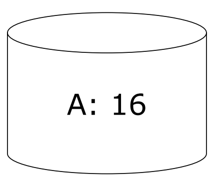
$

Solution

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



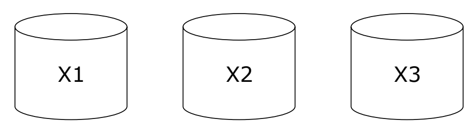
Media Failure (loss of non-volatile storage)



Solution: Make copies of data!

Example 1: Triple Modular Redundancy

- Keep 3 copies on separate disks
- Output(X) --> three outputs
- Input(X) --> three inputs + vote

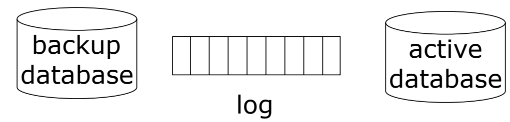


Example 2: Redundant Writes, Single Reads

- Keep N copies on separate disks
- Output(X) --> N outputs
- Input(X) --> Input one copy
  - if ok, done
  - else try another one

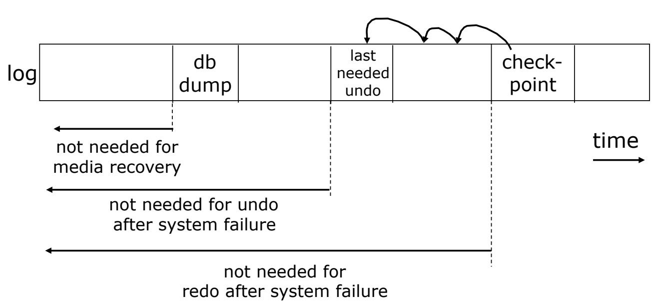
Assumes bad data can be detected

Example 3: DB Dump + Log



- If active database is lost,
  - restore active database from backup
  - bring up-to-date using redo entries in log

When Can Log Be Discarded?



Summary

- Consistency of data
- One source of problems: failures
  - Logging
  - Redundancy
- Another source of problems:
  Data Sharing… next