

CS 277: Database System Implementation

Notes 08: Failure Recovery

Arthur Keller

CS 277 – Spring 2002

Notes 08

1

PART II

- Crash recovery (2 lectures) Ch.17
- Concurrency control (3 lectures) Ch.18
- Transaction processing (2 lects) Ch.19
- Information integration (1 lect) Ch.20

CS 277 – Spring 2002

Notes 08

2

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

CS 277 – Spring 2002

Notes 08

3

Integrity or consistency constraints

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

CS 277 – Spring 2002

Notes 08

4

Definition:

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

CS 277 – Spring 2002

Notes 08

5

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

CS 277 – Spring 2002

Notes 08

6

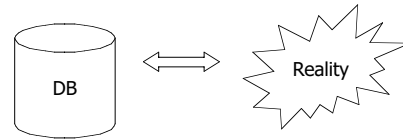
Note: could be "emulated" by simple constraints, e.g.,

account

Acct #	balance	deleted?
--------	------	---------	----------

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

Example 2 Database should reflect real world



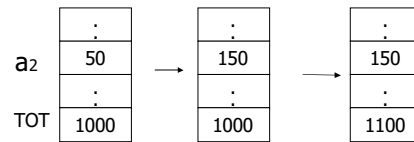
?in any case, continue with constraints...

Observation: DB cannot be consistent always!

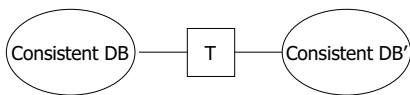
Example: $a_1 + a_2 + \dots + a_n = \text{TOT}$ (constraint)
 Deposit \$100 in a_2 : $\begin{cases} a_2 \leftarrow a_2 + 100 \\ \text{TOT} \leftarrow \text{TOT} + 100 \end{cases}$

Example: $a_1 + a_2 + \dots + a_n = \text{TOT}$ (constraint)

Deposit \$100 in a_2 : $a_2 \leftarrow a_2 + 100$
 $\text{TOT} \leftarrow \text{TOT} + 100$



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 \Rightarrow systems analysts

How can we prevent/fix violations?

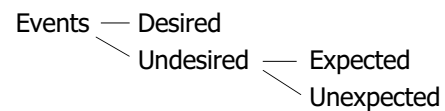
- Chapter 17: due to failures only
- Chapter 18: due to data sharing only
- Chapter 19: due to failures and sharing

Will not consider:

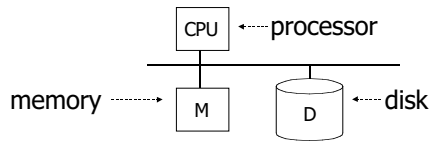
- How to write correct transactions
- How to write correct DBMS
- Constraint checking & repair
That is, solutions studied here do not need to know constraints

Chapter 17 Recovery

- First order of business:
Failure Model



Our failure model



CS 277 – Spring 2002

Notes 08

19

Desired events: see product manuals....

Undesired expected events:

System crash

- memory lost
- cpu halts, resets

that's it!!

Undesired Unexpected: Everything else!

CS 277 – Spring 2002

Notes 08

20

Undesired Unexpected: Everything else!

Examples:

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

CS 277 – Spring 2002

Notes 08

21

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

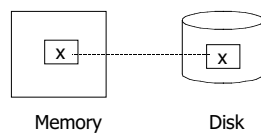
CS 277 – Spring 2002

Notes 08

22

Second order of business:

Storage hierarchy



CS 277 – Spring 2002

Notes 08

23

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

CS 277 – Spring 2002

Notes 08

24

Key problem Unfinished transaction

Example Constraint: $A=B$
 $T_1: A \leftarrow A \times 2$
 $B \leftarrow B \times 2$

CS 277 – Spring 2002 Notes 08 25

$T_1: \text{Read}(A,t); t \leftarrow t \times 2$
 $\text{Write}(A,t);$
 $\text{Read}(B,t); t \leftarrow t \times 2$
 $\text{Write}(B,t);$
 ~~$\text{Output}(A);$~~ failure!
 $\text{Output}(B);$

memory disk

CS 277 – Spring 2002 Notes 08 26

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

CS 277 – Spring 2002 Notes 08 27

One solution: undo logging (immediate modification)

due to: Hansel and Gretel, 782 AD

- Improved in 784 AD to durable undo logging

CS 277 – Spring 2002 Notes 08 28

Undo logging (Immediate modification)

$T_1: \text{Read}(A,t); t \leftarrow t \times 2$ $A=B$
 $\text{Write}(A,t);$
 $\text{Read}(B,t); t \leftarrow t \times 2$
 $\text{Write}(B,t);$
 $\text{Output}(A);$
 $\text{Output}(B);$

memory disk log

CS 277 – Spring 2002 Notes 08 29

One "complication"

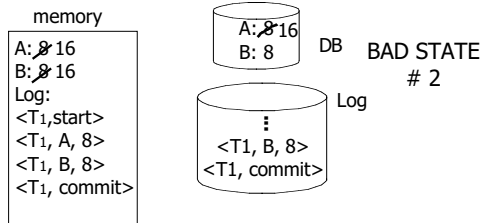
- Log is first written in memory
- Not written to disk on every action

memory DB BAD STATE # 1 Log

CS 277 – Spring 2002 Notes 08 30

One "complication"

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



CS 277 – Spring 2002

Notes 08

31

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

CS 277 – Spring 2002

Notes 08

32

Recovery rules: Undo logging

- For every T_i with $\langle T_i, \text{start} \rangle$ in log:
 - If $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$ in log, do nothing
 - Else { For all $\langle T_i, X, v \rangle$ in log:
 - write (X, v)
 - output (X)
 Write $\langle T_i, \text{abort} \rangle$ to log
- {IS THIS CORRECT??

CS 277 – Spring 2002

Notes 08

33

Recovery rules: Undo logging

- (1) Let S = set of transactions with $\langle T_i, \text{start} \rangle$ in log, but no $\langle T_i, \text{commit} \rangle$ (or $\langle T_i, \text{abort} \rangle$) record in log
- (2) For each $\langle T_i, X, v \rangle$ in log, in reverse order (latest \rightarrow earliest) do:
 - if $T_i \in S$ then {
 - write (X, v)
 - output (X)
- (3) For each $T_i \in S$ do
 - write $\langle T_i, \text{abort} \rangle$ to log

CS 277 – Spring 2002

Notes 08

34

What if failure during recovery?

No problem! / Undo idempotent

CS 277 – Spring 2002

Notes 08

35

To discuss:

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

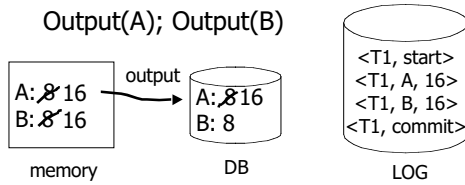
CS 277 – Spring 2002

Notes 08

36

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)



CS 277 – Spring 2002

Notes 08

37

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

CS 277 – Spring 2002

Notes 08

38

Recovery rules: Redo logging

- For every T_i with $\langle T_i, \text{commit} \rangle$ in log:
 - For all $\langle T_i, X, v \rangle$ in log:
 - Write(X, v)
 - Output(X)
- } IS THIS CORRECT??

CS 277 – Spring 2002

Notes 08

39

Recovery rules: Redo logging

- (1) Let $S =$ set of transactions with $\langle T_i, \text{commit} \rangle$ in log
- (2) For each $\langle T_i, X, v \rangle$ in log, in forward order (earliest \rightarrow latest) do:
 - if $T_i \in S$ then
 - Write(X, v)
 - Output(X) ← optional

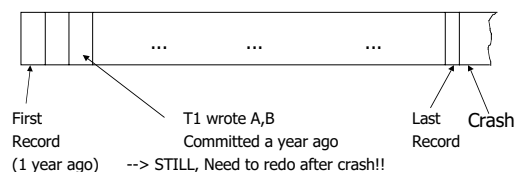
CS 277 – Spring 2002

Notes 08

40

Recovery is very, very **SLOW !**

Redo log:



CS 277 – Spring 2002

Notes 08

41

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

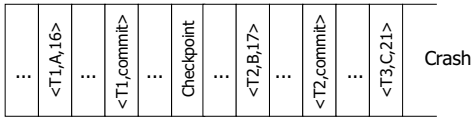
CS 277 – Spring 2002

Notes 08

42

Example: what to do at recovery?

Redo log (disk):



Key drawbacks:

- *Undo logging*: cannot bring backup DB copies up to date
- *Redo logging*: need to keep all modified blocks in memory until commit

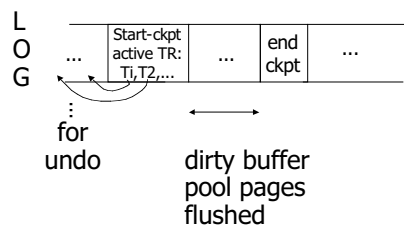
Solution: undo/redo logging!

Update \Rightarrow $\langle T_i, Xid, \text{New } X \text{ val}, \text{Old } X \text{ val} \rangle$
page X

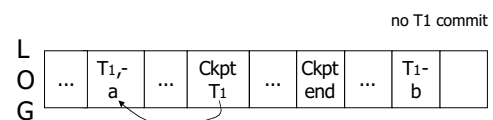
Rules

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

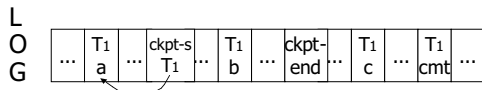
Non-quiet checkpoint



Examples what to do at recovery time?



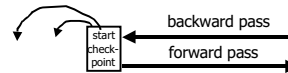
Example



} Redo T1: (redo b,c)

Recovery process:

- **Backwards pass** (end of log \leftarrow 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 \leftarrow end of log)
 - redo actions of S transactions



Real world actions

E.g., dispense cash at ATM

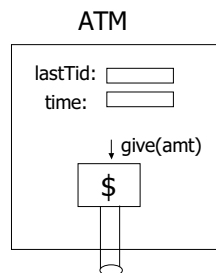
$T_i = a_1 a_2 \dots a_j \dots a_n$

↓
\$

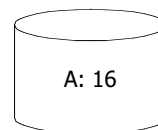
Solution

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

Give\$\$
(amt, Tid, time)



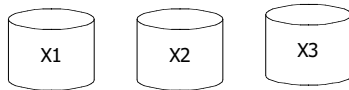
Media failure (loss of non-volatile storage)



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



CS 277 – Spring 2002

Notes 08

55

Example #2 Redundant writes, Single reads

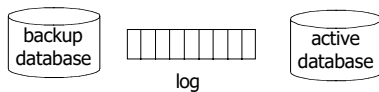
- Keep N copies on separate disks
 - Output(X) --> N outputs
 - Input(X) --> Input one copy
 - if ok, done
 - else try another one
- Assumes bad data can be detected

CS 277 – Spring 2002

Notes 08

56

Example #3: DB Dump + Log



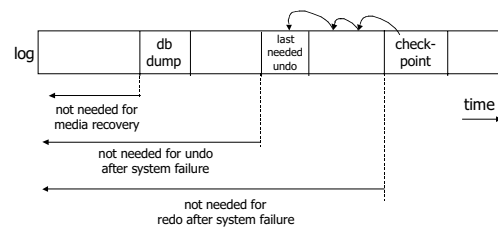
- If active database is lost,
 - restore active database from backup
 - bring up-to-date using redo entries in log

CS 277 – Spring 2002

Notes 08

57

When can log be discarded?



CS 277 – Spring 2002

Notes 08

58

Summary

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

CS 277 – Spring 2002

Notes 08

59