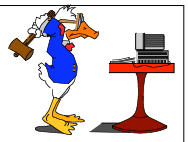
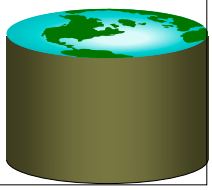
Crash Recovery, Part 2



R&G - Chapter 18

If you are going to be in the logging business, one of the things that you have to do is to learn about heavy equipment.

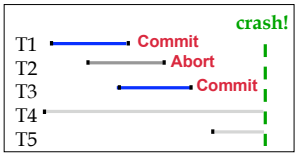
Robert VanNatta,
Logging History of Columbia County



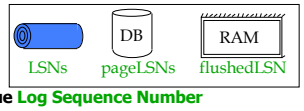
Motivation

- **Atomicity:**
  - Transactions may abort ("Rollback").
- **Durability:**
  - What if DBMS stops running? (Causes?)

❖ Desired state after system restarts:
- T1 & T3 should be durable.
- T2, T4 & T5 should be aborted (effects not seen).

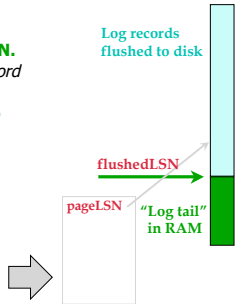


WAL & the Log



- **Each log record has a unique Log Sequence Number (LSN).**
  - LSNs always increasing.
- **Each *data page* contains a pageLSN.**
  - The LSN of the most recent *log record* for an update to that page.
- **System keeps track of flushedLSN.**
  - The max LSN flushed so far.
- **WAL: For a page i to be written must flush log at least to the point where:**

$pageLSN_i \le flushedLSN$



Log Records

LogRecord fields:

LSN
prevLSN
XID
type
update records only:
- pageID
- length
- offset
- before-image
- after-image

prevLSN is the LSN of the previous log record written by *this* Xact (so records of an Xact form a linked list backwards in time)

Possible log record types:
- Update, Commit, Abort
- Checkpoint **(for log maintenance)**
- **Compensation Log Records (CLRs)**
  - for UNDO actions
- End **(end of commit or abort)**

Other Log-Related State

- **Two in-memory tables:**
- **Transaction Table**
  - One entry per currently active Xact.
    - entry removed when Xact commits or aborts
  - Contains XID, status (running/committing/aborting), and lastLSN (most recent LSN written by Xact).
- **Dirty Page Table:**
  - One entry per dirty page currently in buffer pool.
  - Contains recLSN -- the LSN of the log record which ***first*** caused the page to be dirty.

The Big Picture: What's Stored Where

LogRecords
prevLSN
XID
type
pageID
length
offset
before-image
after-image

Data pages
each with a pageLSN

master record
LSN of most recent checkpoint

Xact Table
lastLSN
status

Dirty Page Table
recLSN

flushedLSN

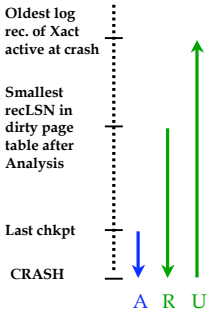
Normal Execution of an Xact

- **Series of reads & writes, followed by commit or abort.**
  - We will assume that disk write is atomic.
    - In practice, additional details to deal with non-atomic writes.
- **Strict 2PL.**
- **STEAL, NO-FORCE buffer management, with Write-Ahead Logging.**

Checkpointing

- **Conceptually, keep log around for all time. Obviously this has performance/implementation problems...**
- **Periodically, the DBMS creates a checkpoint, in order to minimize crash recovery time.**
- **Write to log:**
  - begin_checkpoint record: Indicates when chkpt began.
  - end_checkpoint record: Contains current *Xact table* and *dirty page table*. This is a `fuzzy checkpoint':
    - Other Xacts continue to run; so these tables accurate only as of the time of the begin_checkpoint record.
    - No attempt to force dirty pages to disk; effectiveness of checkpoint limited by oldest unwritten change to a dirty page.
- **Store LSN of most recent checkpoint record in a safe place (*master record*).**

Crash Recovery: Big Picture



- Start from a checkpoint (found via master record).
- Three phases. Need to:
  - Analysis - Figure out which Xacts committed since checkpoint, which failed.
  - REDO *all* actions. (repeat history)
  - UNDO effects of failed Xacts.

Recovery: The Analysis Phase

- **Re-establish knowledge of state at checkpoint.**
  - via transaction table and dirty page table stored in the checkpoint
- **Scan log forward from checkpoint.**
  - End record: Remove Xact from Xact table.
  - Other records: Add Xact to Xact table, set lastLSN=LSN, (update Xact status).
  - also, for Update records: If page P not in Dirty Page Table, Add P to DPT, set its recLSN=LSN.
- **At end of Analysis...**
  - transaction table says which xacts were active at time of crash.
  - DPT says which dirty pages *might not* have made it to disk

Phase 2: The REDO Phase

- **We *repeat History* to reconstruct state at crash:**
  - Reapply *all* updates (even of aborted Xacts!), redo CLRs.
- **Scan forward from log rec containing smallest recLSN in DPT.** Q: why start here?
- **For each update log record or CLR with a given LSN, REDO the action unless:**
  - Affected page is not in the Dirty Page Table, or
  - Affected page is in D.P.T., but has recLSN > LSN, or
  - pageLSN (in DB) ≥ LSN. (this last case requires I/O)
- **To REDO an action:**
  - Reapply logged action.
  - Set pageLSN to LSN. No additional logging, no forcing!

Phase 3: The UNDO Phase

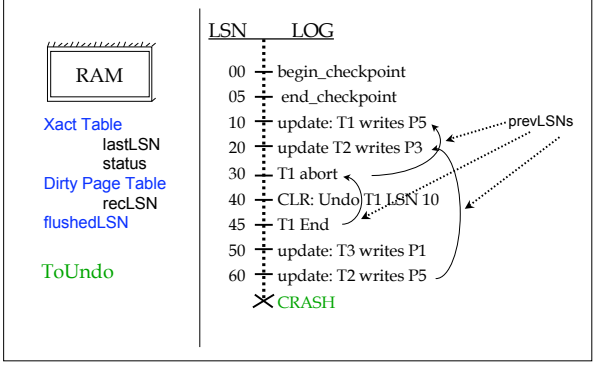
ToUndo={lastLSNs of all Xacts in the Trans Table}

Repeat:

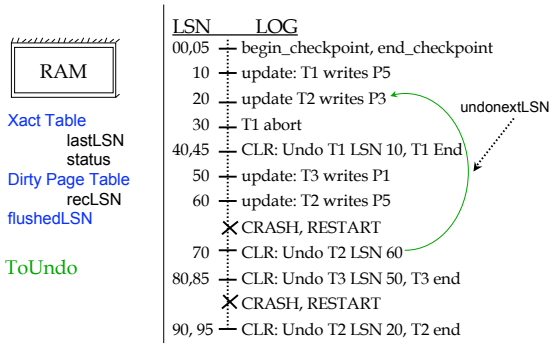
- Choose (and remove) largest LSN among ToUndo.
- If this LSN is a CLR and undonextLSN==NULL
  - Write an End record for this Xact.
- If this LSN is a CLR, and undonextLSN != NULL
  - Add undonextLSN to ToUndo
- Else this LSN is an update. Undo the update, write a CLR, add prevLSN to ToUndo.

Until ToUndo is empty.

Example of Recovery



Example: Crash During Restart!



Additional Crash Issues

- **What happens if system crashes during Analysis? During REDO?**
- **How do you limit the amount of work in REDO?**
  - Flush asynchronously in the background.
- **How do you limit the amount of work in UNDO?**
  - Avoid long-running Xacts.

Summary of Logging/Recovery

- **Recovery Manager guarantees Atomicity & Durability.**
- **Use WAL to allow STEAL/NO-FORCE w/o sacrificing correctness.**
- **LSNs identify log records; linked into backwards chains per transaction (via prevLSN).**
- **pageLSN allows comparison of data page and log records.**

Summary, Cont.

- **Checkpointing: A quick way to limit the amount of log to scan on recovery.**
- **Recovery works in 3 phases:**
  - Analysis: Forward from checkpoint.
  - Redo: Forward from oldest recLSN.
  - Undo: Backward from end to first LSN of oldest Xact alive at crash.
- **Upon Undo, write CLRs.**
- **Redo "repeats history": Simplifies the logic!**