Crash Recovery

From Chapter 16, 18
Review: The ACID properties

- **Atomicity:** All actions in a Xact happen, or none happen
- **Consistency:** Each Xact transforms the database from one consistent state to another
- **Isolation:** Execution of concurrent transactions is as though they are evaluated in some serial order
- **Durability:** If a Xact commits, its effects persist
Motivation

- **Atomicity:**
  - Transactions may abort ("Rollback").

- **Durability:**
  - What if DBMS stops running?
    (Causes?)

```
T1  T2  T3  T4  T5
```

```
crash!
```
Handling the Buffer Pool

- **Force** every write to disk?

- **Steal** buffer-pool frames from uncommitted Xacts?
Basic Idea: Logging

- Record REDO and UNDO information, for every update, in a log.

- Log: An ordered list of REDO/UNDO actions
Write-Ahead Logging (WAL)

- The Write-Ahead Logging Protocol:
  1.
  2.

- #1 guarantees Atomicity.
- #2 guarantees Durability.
The Big Picture: What’s Stored Where

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

Data pages
- each with a pageLSN

Dirty Page Table
- recLSN

Xact Table
- lastLSN
- status

flushedLSN
Transaction Commit

- Write **commit** record to log.
- All log records up to Xact’s **lastLSN** are flushed.

- Commit() returns.
- Write **end** record to log.
Simple Transaction Abort

- Get lastLSN of Xact from Xact table.
- Can follow chain of log records backward via the prevLSN field.
- Before starting UNDO, write an *Abort* log record.

- Before restoring old value of a page, write a CLR (undonextLSN = prevLSN of undone record)
- At end of UNDO, write an “end” log record.
Crash Recovery: Big Picture

- Start from a checkpoint
- Three phases.
  - Analysis
  - REDO
  - UNDO
Recovery: The Analysis Phase

- Reconstruct state at checkpoint.
  - via end_checkpoint record.
- Scan log forward from checkpoint.
  - End record: Remove Xact from Xact table.
  - Other records: Add Xact to Xact table, set lastLSN=LSN, change Xact status on commit.
  - Update record: If P not in Dirty Page Table,
    - Add P to D.P.T., set its recLSN=LSN.
Recovery: 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 D.P.T. For each CLR or update log rec 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.
- **To REDO** an action:
  - Reapply logged action.
  - Set pageLSN to LSN. No additional logging!
Recovery: The UNDO Phase

ToUndo={ / | / a lastLSN of a “loser” Xact}

Repeat:

- Choose 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
- In this LSN is an Abort, add prevLSN to ToUndo.
- Else this LSN is an update. Undo the update, write a CLR, and
  - if (prevLSN != NULL ) add prevLSN to ToUndo
  - Else write an End record for this Xact

Until ToUndo is empty.
ICE: Example of Recovery

LSN   LOG
00    begin_checkpoint
05    end_checkpoint
10    update: T1 writes P5
20    update: T2 writes P3
30    T1 abort
40    CLR: Undo T1 LSN 10
45    T1 End
50    update: T3 writes P1
60    update: T2 writes P5

CRASH, RESTART

RAM

Xact Table
lastLSN
status
Dirty Page Table
recLSN
flushedLSN
ToUndo

prevLSNs
Summary

- Recovery manager ensures Atomicity and Durability
- Logging is used
- Write-Ahead-Logging
- Checkpoints
- ARIES: Analysis, REDO, UNDO