Chapter 19: Recovery System

Database System Concepts, 7th Ed.

Outline

- Failure Classification
- Storage Structure
- Recovery and Atomicity
- Log-Based Recovery
- Remote Backup Systems

Failure Classification

- **Transaction failure** :
  - **Logical errors**: transaction cannot complete due to some internal error condition
  - **System errors**: the database system must terminate an active transaction due to an error condition (e.g., deadlock)
- **System crash**: a power failure or other hardware or software failure causes the system to crash.
  - **Fail-stop assumption**: non-volatile storage contents are assumed to not be corrupted by system crash
    - Database systems have numerous integrity checks to prevent corruption of disk data
- **Disk failure**: a head crash or similar disk failure destroys all or part of disk storage
  - Destruction is assumed to be detectable: disk drives use checksums to detect failures

Recovery Algorithms

- Suppose transaction T_i transfers \$50 from account A to account B
  - Two updates: subtract 50 from A and add 50 to B
- Transaction T_i requires updates to A and B to be output to the database.
  - A failure may occur after one of these modifications have been made but before both of them are made.
  - Modifying the database without ensuring that the transaction will commit may leave the database in an inconsistent state
  - Not modifying the database may result in lost updates if failure occurs just after transaction commits
- Recovery algorithms have two parts
  1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures
  2.

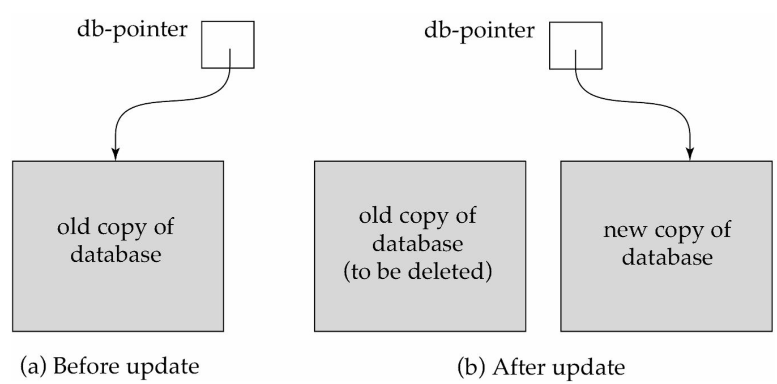
Storage Structure

- **Volatile storage**:
  - Does not survive system crashes
  - Examples: main memory, cache memory
- **Nonvolatile storage**:
  - Survives system crashes
  - Examples: disk, tape, flash memory, non-volatile RAM
  - But may still fail, losing data
- **Stable storage**:
  - A mythical form of storage that survives all failures
  - Approximated by maintaining multiple copies on distinct nonvolatile media
  - See book for more details on how to implement stable storage

Recovery and Atomicity

- To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself.
- We study **log-based recovery mechanisms** in detail
  - We first present key concepts
  - And then present the actual recovery algorithm
- Less used alternative: **shadow-copy** and **shadow-paging** (brief details in book)

shadow-copy



Log-Based Recovery

- A **log** is a sequence of **log records**. The records keep information about update activities on the database.
  - The **log** is kept on stable storage
- When transaction T_i starts, it registers itself by writing a

  <T_i **start**> log record

- *Before* T_i executes **write**(X), a log record

  <T_i, X, V_1, V_2>

  is written, where V_1 is the value of X before the write (the **old value**), and V_2 is the value to be written to X (the **new value**).
- When T_i finishes it last statement, the log record <T_i **commit**> is written.
- Two approaches using logs
  - Immediate database modification
  - Deferred database modification.

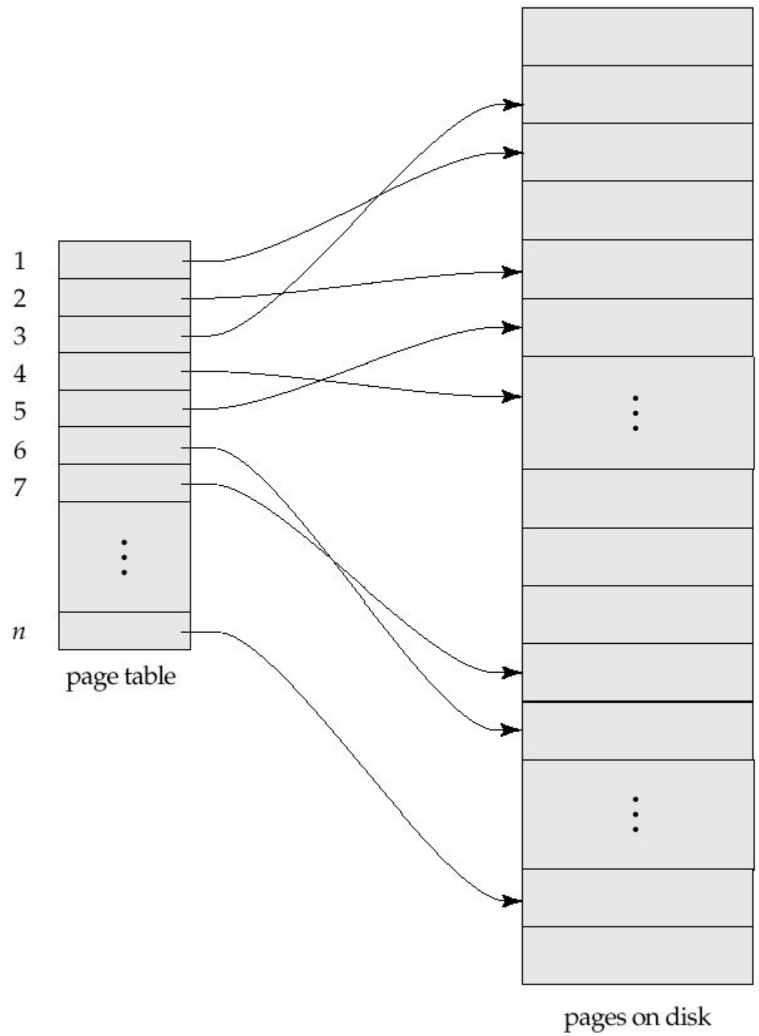
Extra

Database System Concepts, 7th Ed.

Shadow Paging

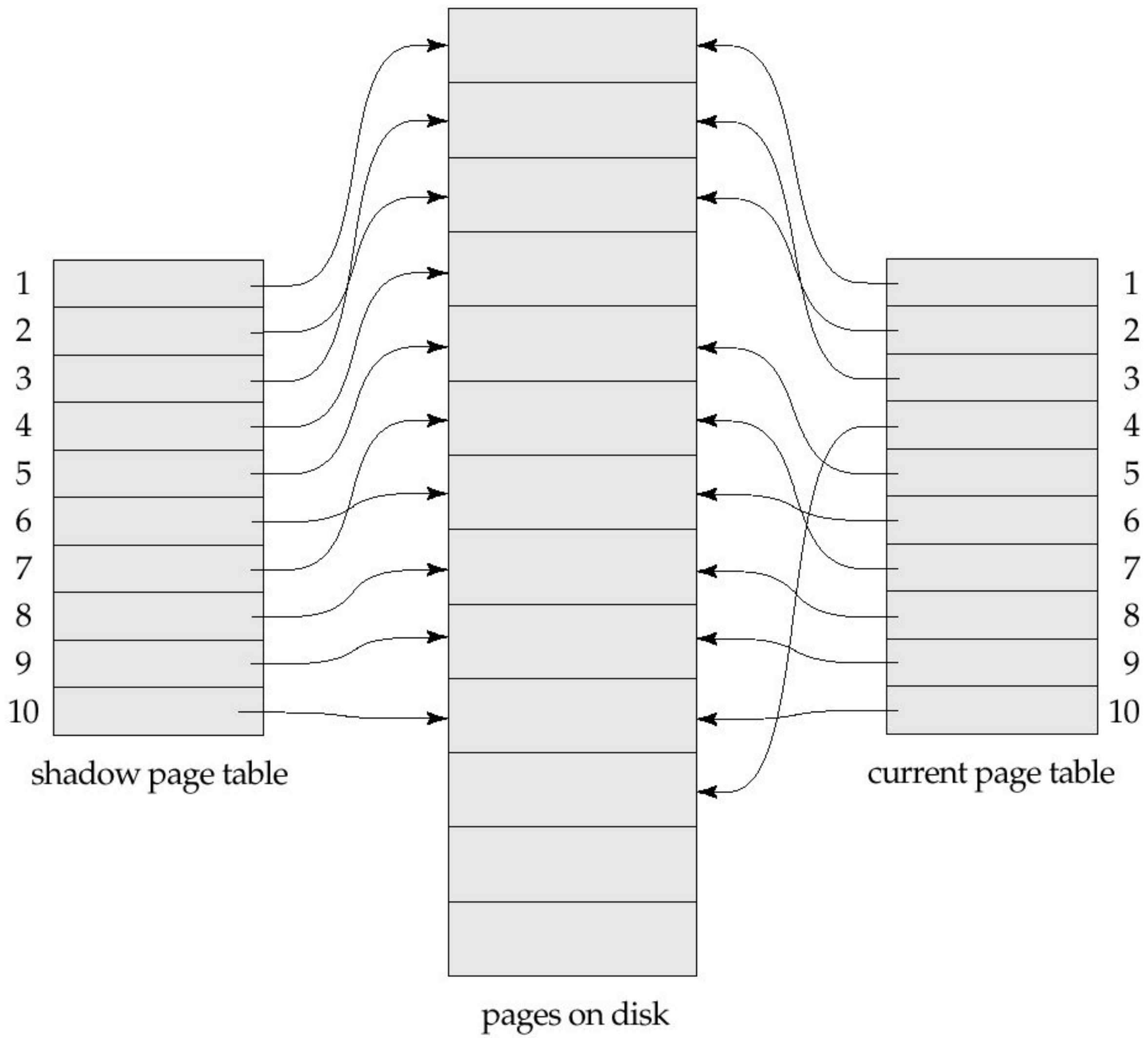
- **Shadow paging** is an alternative to log-based recovery; this scheme is useful if transactions execute serially
- Idea: maintain *two* page tables during the lifetime of a transaction –the **current page table**, and the **shadow page table**
- Store the shadow page table in nonvolatile storage, such that state of the database prior to transaction execution may be recovered.
  - Shadow page table is never modified during execution
- To start with, both the page tables are identical. Only current page table is used for data item accesses during execution of the transaction.
- Whenever any page is about to be written for the first time
  - A copy of this page is made onto an unused page.
  - The current page table is then made to point to the copy
  - The update is performed on the copy

Sample Page Table



Example of Shadow Paging

Shadow and current page tables after write to page 4



Shadow Paging (Cont.)

- To commit a transaction :

1. Flush all modified pages in main memory to disk
2. Output current page table to disk
3. Make the current page table the new shadow page table, as follows:
    - keep a pointer to the shadow page table at a fixed (known) location on disk.
    - to make the current page table the new shadow page table, simply update the pointer to point to current page table on disk

- Once pointer to shadow page table has been written, transaction is committed.
- No recovery is needed after a crash — new transactions can start right away, using the shadow page table.
- Pages not pointed to from current/shadow page table should be freed (garbage collected).