Database Recovery Techniques

Dr. Sumeet Kaur Sehra

Introduction

- Recovery algorithms
- Recovery concepts
  - Write-ahead logging
  - In-place versus shadow updates
  - Rollback
  - Deferred update
  - Immediate update
- Certain recovery techniques best used with specific concurrency control methods

Adapted from Navathe (7th Edition)

Recovery Concepts

- Recovery process restores database to most recent consistent state before time of failure
- Information kept in system log
- Typical recovery strategies
  - Restore backed-up copy of database
    - Best in cases of extensive damage
  - Identify any changes that may cause inconsistency
    - Best in cases of noncatastrophic failure
    - Some operations may require redo

Adapted from Navathe (7th Edition)

Recovery Concepts (cont’d.)

Deferred update techniques

- Do not physically update the database until after transaction commits
- Undo is not needed; redo may be needed

Immediate update techniques

- Database may be updated by some operations of a transaction before it reaches commit point
- Operations also recorded in log
- Recovery still possible

Adapted from Navathe (7th Edition)

Recovery Concepts (cont’d.)

Undo and redo operations required to be idempotent

- Executing operations multiple times equivalent to executing just once
- Entire recovery process should be idempotent

Caching (buffering) of disk blocks

- DBMS cache: a collection of in-memory buffers
- Cache directory keeps track of which database items are in the buffers

Adapted from Navathe (7th Edition)

Recovery Concepts (cont’d.)

Cache buffers replaced (flushed) to make space for new items

Dirty bit associated with each buffer in the cache

- Indicates whether the buffer has been modified

Contents written back to disk before flush if dirty bit equals one

Pin-unpin bit

- Page is pinned if it cannot be written back to disk yet

Adapted from Navathe (7th Edition)

Recovery Concepts (cont’d.)

Main strategies

- In-place updating
  - Writes the buffer to the same original disk location
  - Overwrites old values of any changed data items
- Shadowing
  - Writes an updated buffer at a different disk location, to maintain multiple versions of data items
  - Not typically used in practice

Before-image: old value of data item

After-image: new value of data item

Adapted from Navathe (7th Edition)

Recovery Concepts (cont’d.)

Write-ahead logging

- Ensure the before-image (BFIM) is recorded
- Appropriate log entry flushed to disk
- Necessary for UNDO operation if needed

UNDO-type log entries

REDO-type log entries

Adapted from Navathe (7th Edition)

Recovery Concepts (cont’d.)

Steal/no-steal and force/no-force

Specify rules that govern when a page from the database cache can be written to disk

No-steal approach

Cache buffer page updated by a transaction cannot be written to disk before the transaction commits

Steal approach

Recovery protocol allows writing an updated buffer before the transaction commits

Adapted from Navathe (7th Edition)

Recovery Concepts (cont'd.)

Force approach

All pages updated by a transaction are immediately written to disk before the transaction commits

Otherwise, no-force

Typical database systems employ a steal/no-force strategy

Avoids need for very large buffer space

Reduces disk I/O operations for heavily updated pages

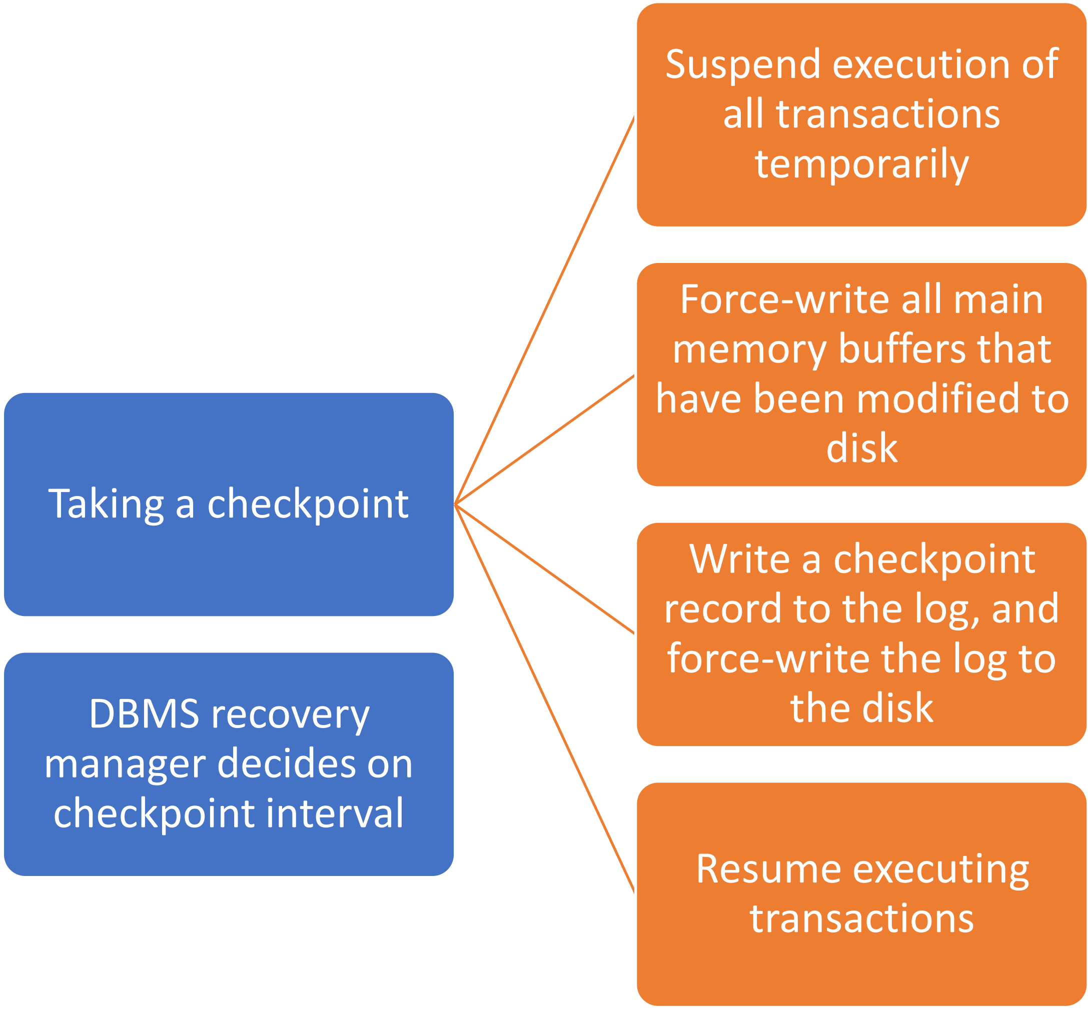
Adapted from Navathe (7th Edition)

Recovery Concepts (cont’d.)

- Write-ahead logging protocol for recovery algorithm requiring both UNDO and REDO
  - BFIM of an item cannot be overwritten by its after image until all UNDO-type log entries have been force-written to disk
  - Commit operation of a transaction cannot be completed until all REDO-type and UNDO-type log records for that transaction have been force-written to disk

Adapted from Navathe (7th Edition)

Checkpoints in the System Log and Fuzzy Checkpointing



Taking a checkpoint

- Suspend execution of all transactions temporarily
- Force-write all main memory buffers that have been modified to disk
- Write a checkpoint record to the log, and force-write the log to the disk
- Resume executing transactions

DBMS recovery manager decides on checkpoint interval

Adapted from Navathe (7th Edition)

Checkpoints in the System Log and Fuzzy Checkpointing (cont'd.)

- Fuzzy checkpointing
  - System can resume transaction processing after a begin_checkpoint record is written to the log
  - Previous checkpoint record maintained until end_checkpoint record is written

Transaction Rollback

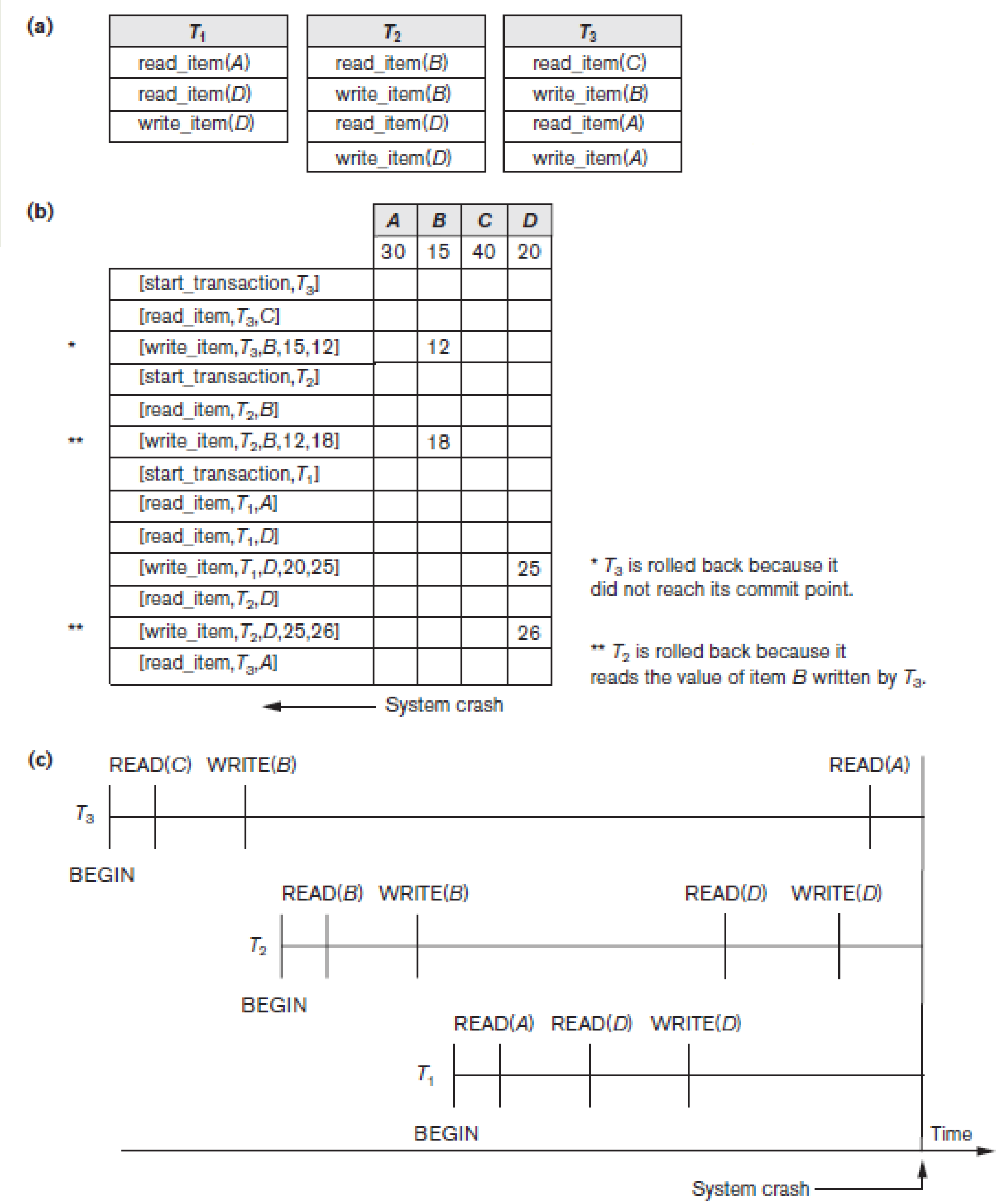
Transaction failure after update but before commit

- Necessary to roll back the transaction
- Old data values restored using undo-type log entries

Cascading rollback

- If transaction T is rolled back, any transaction S that has read value of item written by T must also be rolled back
- Almost all recovery mechanisms designed to avoid this

Illustrating cascading rollback (a process that never occurs
in strict or cascadeless schedules) (a) The read and write operations of three transactions (b) System log at point of crash (c) Operations before the crash



(a)

T_1	T_2	T_3
read_item(A)	read_item(B)	read_item(C)
read_item(D)	write_item(B)	write_item(B)
write_item(D)	read_item(D)	read_item(A)
	write_item(D)	write_item(A)

(b)

		A	B	C	D
		30	15	40	20
	[start_transaction,T_3]				
	[read_item,T_3,C]				
*	[write_item,T_3,B,15,12]		12		
	[start_transaction,T_2]				
	[read_item,T_2,B]				
**	[write_item,T_2,B,12,18]		18		
	[start_transaction,T_1]				
	[read_item,T_1,A]				
	[read_item,T_1,D]				
	[write_item,T_1,D,20,25]				25
	[read_item,T_2,D]				
**	[write_item,T_2,D,25,26]				26
	[read_item,T_3,A]				

← System crash

* T_3 is rolled back because it did not reach its commit point.

** T_2 is rolled back because it reads the value of item B written by T_3.

(c)

Adapted from Navathe (7th Edition)

Transactions that Do Not Affect the Database

Example actions: generating and printing messages and reports

If transaction fails before completion, may not want user to get these reports

- Reports should be generated only after transaction reaches commit point

Commands that generate reports issued as batch jobs executed only after transaction reaches commit point

- Batch jobs canceled if transaction fails

Adapted from Navathe (7th Edition)

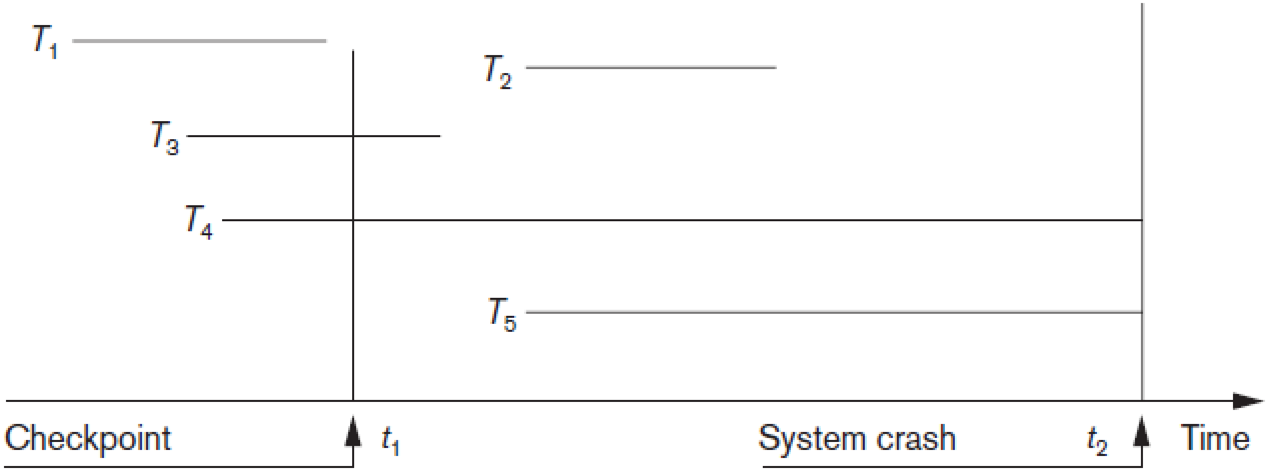
NO-UNDO/REDO Recovery Based on Deferred Update

- Deferred update concept
  - Postpone updates to the database on disk until the transaction completes successfully and reaches its commit point
  - Redo-type log entries are needed
  - Undo-type log entries not necessary
  - Can only be used for short transactions and transactions that change few items
    - Buffer space an issue with longer transactions

NO-UNDO/REDO Recovery Based on Deferred Update (cont'd.)

- Deferred update protocol
  - Transaction cannot change the database on disk until it reaches its commit point
    - All buffers changed by the transaction must be pinned until the transaction commits (no-steal policy)
  - Transaction does not reach its commit point until all its REDO-type log entries are recorded in log and log buffer is force-written to disk

NO-UNDO/REDO Recovery Based on Deferred Update (cont’d.)



An example of a recovery timeline to illustrate the effect of checkpointing

Adapted from Navathe (7th Edition)

Recovery Techniques Based on Immediate Update

Database can be updated immediately

No need to wait for transaction to reach commit point

Not a requirement that every update be immediate

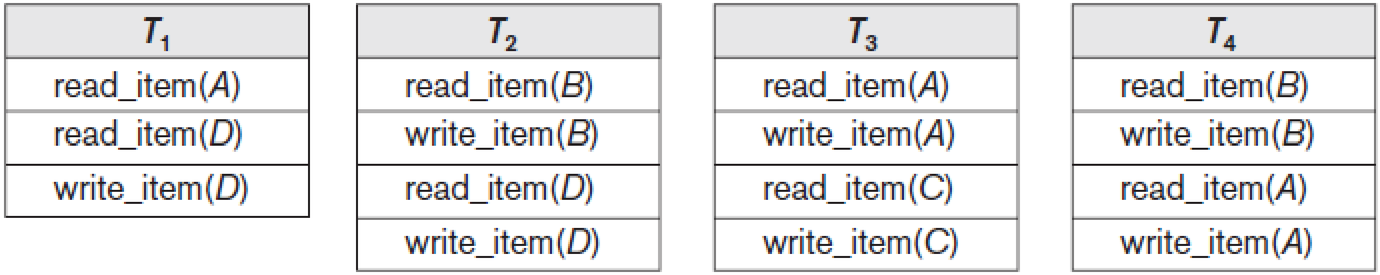
UNDO-type log entries must be stored

Recovery algorithms

UNDO/NO-REDO (steal/force strategy)

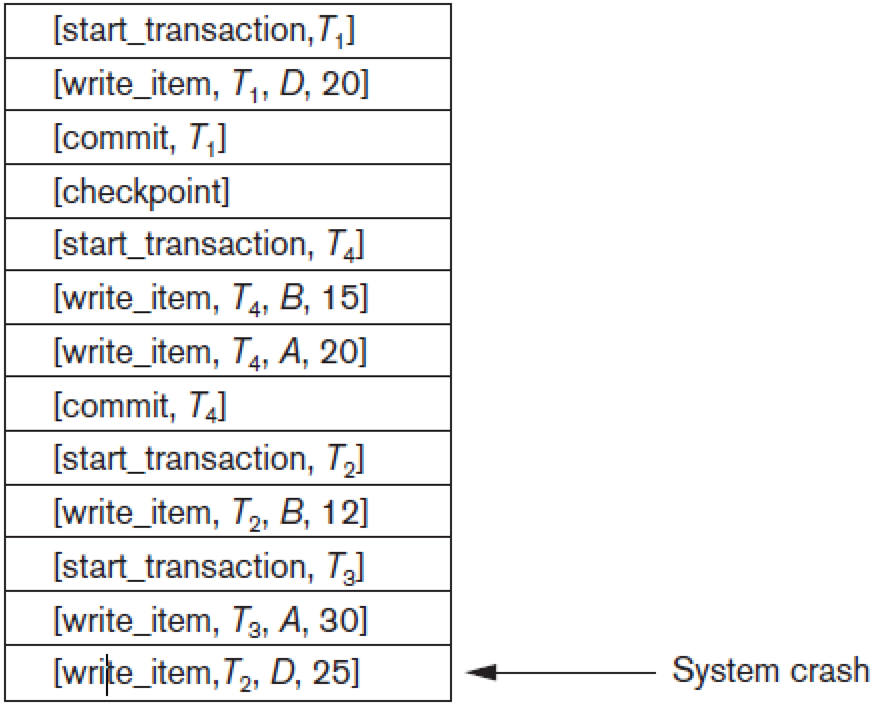
UNDO/REDO (steal/no-force strategy)

Adapted from Navathe (7th Edition)



T_1	T_2	T_3	T_4
read_item(*A*)	read_item(*B*)	read_item(*A*)	read_item(*B*)
read_item(*D*)	write_item(*B*)	write_item(*A*)	write_item(*B*)
write_item(*D*)	read_item(*D*)	read_item(*C*)	read_item(*A*)
	write_item(*D*)	write_item(*C*)	write_item(*A*)

An example of recovery using deferred update with concurrent transactions (a) The READ and WRITE operations of four transactions (b) System log at the point of crash



[start_transaction, T_1]
[write_item, T_1, *D*, 20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, *B*, 15]
[write_item, T_4, *A*, 20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2, *B*, 12]
[start_transaction, T_3]
[write_item, T_3, *A*, 30]
[write_item, T_2, *D*, 25]

System crash

T_2 and T_3 are ignored because they did not reach their commit points.

T_4 is redone because its commit point is after the last system checkpoint.

Adapted from Navathe (7th Edition)

22.4 Shadow Paging

No log required in a single-user environment

- Log may be needed in a multiuser environment for the concurrency control method

Shadow paging considers disk to be made of *n* fixed-size disk pages

- Directory with *n* entries is constructed
- When transaction begins executing, directory copied into shadow directory to save while current directory is being used
- Shadow directory is never modified

Adapted from Navathe (7th Edition)

Shadow Paging (cont'd.)

New copy of the modified page created and stored elsewhere

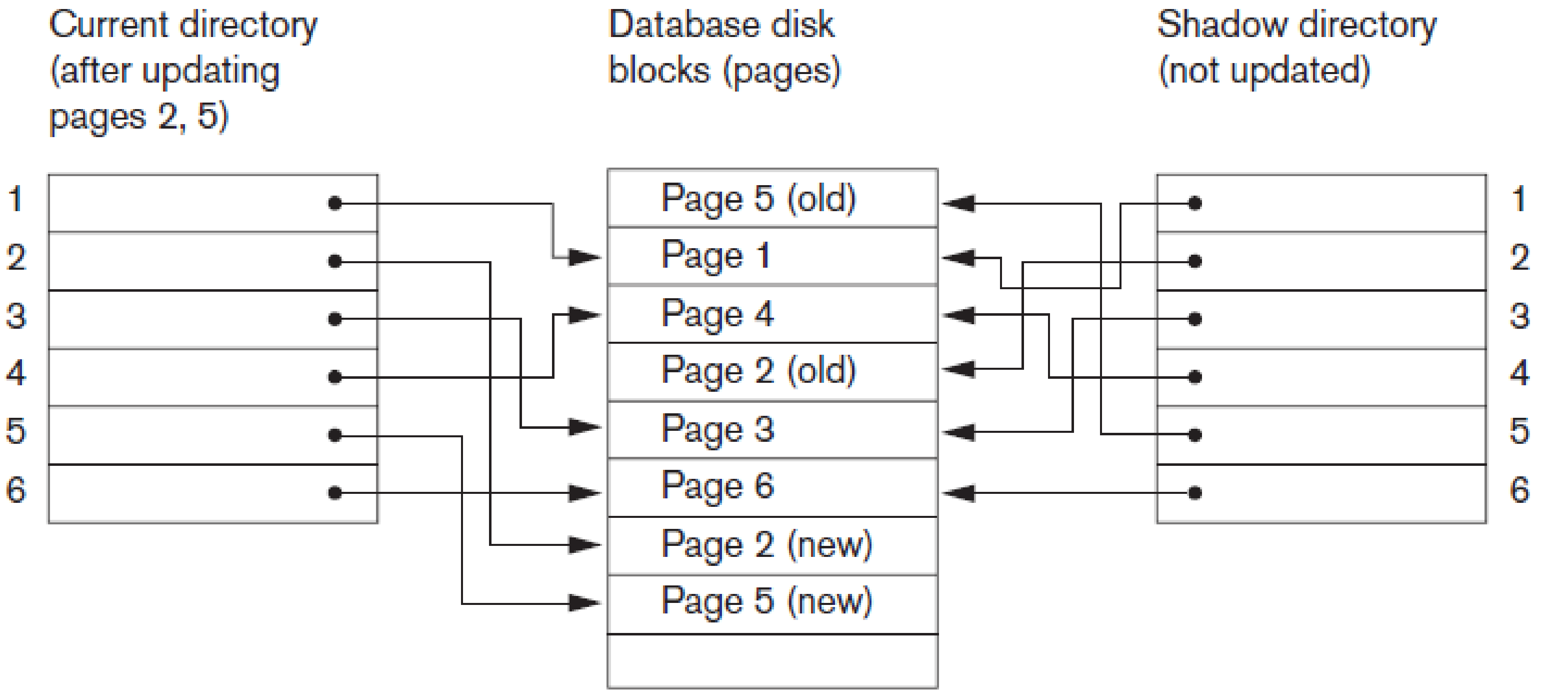
- Current directory modified to point to new disk block
- Shadow directory still points to old disk block

Failure recovery

- Discard current directory
- Free modified database pages
- NO-UNDO/NO-REDO technique

Adapted from Navathe (7th Edition)

Shadow Paging (cont'd.)


Current directory (after updating pages 2, 5)
Database disk blocks (pages)
Shadow directory (not updated)
1
2
3
4
5
6
Page 5 (old)
Page 1
Page 4
Page 2 (old)
Page 3
Page 6
Page 2 (new)
Page 5 (new)
1
2
3
4
5
6

[5] The directory is similar to the page table maintained by the operating system for each process.

An example of shadow paging

Adapted from Navathe (7th Edition)

Recovery in Multidatabase Systems

Two-level recovery mechanism

Global recovery manager (coordinator) needed to maintain recovery information

Coordinator follows two-phase commit protocol

- Phase 1: Prepare for commit message
  - Ready to commit or cannot commit signal returned
- Phase 2: Issue commit signal

Either all participating databases commit the effect of the transaction or none of them do

Adapted from Navathe (7th Edition)

Recovery in Multidatabase Systems (cont’d.)

- Always possible to recover to a state where either the transaction is committed or it is rolled back
- Failure during phase 1 requires rollback
- Failure during phase 2 means successful transaction can recover and commit

Adapted from Navathe (7th Edition)

Database Backup and Recovery from Catastrophic Failures

Database backup

- Entire database and log periodically copied onto inexpensive storage medium
- Latest backup copy can be reloaded from disk in case of catastrophic failure

Backups often moved to physically separate locations

- Subterranean storage vaults

Adapted from Navathe (7th Edition)

Database Backup and Recovery from Catastrophic Failures (cont'd.)

- Backup system log at more frequent intervals and copy to magnetic tape
  - System log smaller than database
    - Can be backed up more frequently
  - Benefit: users do not lose all transactions since last database backup

Adapted from Navathe (7th Edition)