Database Recovery Techniques

Dr. Jeevani Goonetillake

Types of Failure

- The database may become unavailable for use due to
  - **Transaction failure**: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
  - **System failure**: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
  - **Media failure**: Disk head crash, power disruption, etc.

Purpose of Database Recovery

– **Recovery** manager is responsible for transaction atomicity and durability.
  - Undo actions of aborted transactions.
  - Actions from committed transactions can survive system crashes.

– To bring the database into the last consistent state, which existed prior to the failure.

Transaction Log

- For recovery from any type of failure data values prior to modification (BFIM - Before Image) and the new value after modification (AFIM – After Image) are required.
- These values and other information is stored in a sequential file called Transaction log. A sample log is given below. Back P and Next P point to the previous and next log records of the same transaction.

T ID	Back P	Next P	Operation	Data item	BFIM	AFIM
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

Data Caching

- Data items to be modified are first stored into database cache by the Cache Manager (CM).

- After modification they are flushed (written) to the disk.

Data Update

- **In-place update**: The disk version of the data item is overwritten by the cache version (i.e. writes the buffer back to the same original disk location).
  - **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
  - **Deferred Update**: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.
- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.

Write-Ahead Logging

- When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging (WAL)** protocol. WAL states that
  – **For Undo**: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).
  – **For Redo**: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

UCSC

BIT

Steal/No-Steal and Force/No-Force

Possible ways for flushing database cache to database disk:

- **Steal**: Cache page updated by a transaction can be flushed to disk before transaction commits.
- **No-Steal**: Cache cannot be flushed before transaction commit.
- **Force**: all Cache pages updated by a transaction are immediately flushed (forced) to disk when the transaction commits.
- **No-Force**: Modified pages may not immediately be written to disk after a transaction commits.

Steal/No-Steal and Force/No-Force

– These give rise to four different ways for handling recovery:
  - Steal/No-Force (Undo/Redo)
  - Steal/Force (Undo/No-redo)
  - No-Steal/No-Force (Redo/No-undo)
  - No-Steal/Force (No-undo/No-redo)

Typical database systems employ a *steal/no_force* strategy.

Transaction **Roll-back (Undo)** and **Roll-Forward (Redo)**

To maintain atomicity, a transaction's operations are redone or undone.

- **Undo**: Restore all BFIMs on to disk (Remove all AFIMs).
- **Redo**: Restore all AFIMs on to disk.

– Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen.

UCSC

BIT

Checkpoints in the System Log

- Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:
  1. Suspend execution of transactions temporarily.
  2. Force write modified buffer data to disk.
  3. Write a [checkpoint] record to the log, save the log to disk.
  4. Resume normal transaction execution.

- During recovery redo or undo is required to transactions appearing after [checkpoint] record.

Deferred Update

- Defer or postpone any actual updates to the database until the transaction completes its execution successfully and reaches its commit point.
- After the transaction reaches its commit point and the log is force-written to disk, the updates are recorded in the database.
- If a transaction fails before reaching its commit point there is no need to undo any operation. Hence this is known as **NO_UNDO/REDO recovery algorithm**.

Deferred Update

(a)

T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

(b)

[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[start_transaction, T_2]
[write_item, T_2, B, 10]
[write_item, T_2, D, 25] ← System crash

The [write_item,...] operations of T_1 are redone.

T_2 log entries are ignored by the recovery process.

Figure 19.2
An example of recovery using deferred update in a single-user environment. (a) The READ and WRITE operations of two transactions. (b) The system log at the point of crash.

UCSC

BIT

Deferred Update

(a)

T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

T_3
read_item(A)
write_item(A)
read_item(C)
write_item(C)

T_4
read_item(B)
write_item(B)
read_item(A)
write_item(A)

(b)

[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.

Figure 19.4
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.

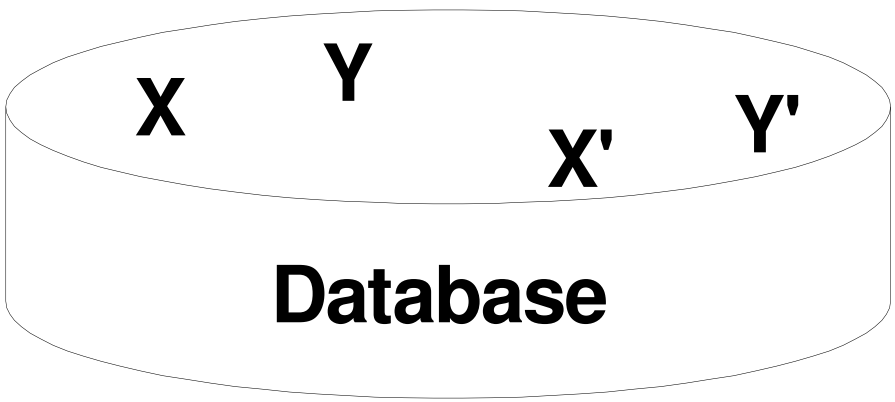
Immediate Update

Undo/No-redo Algorithm

- In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits.
- For this reason the recovery manager **undoes** all transactions during recovery.
- No transaction is **redone**.
- It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

Shadow Paging

- The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.



X and Y: Shadow copies of data items

X' and Y': Current copies of data items

Shadow Paging

During transaction execution, the shadow directory is never modified.

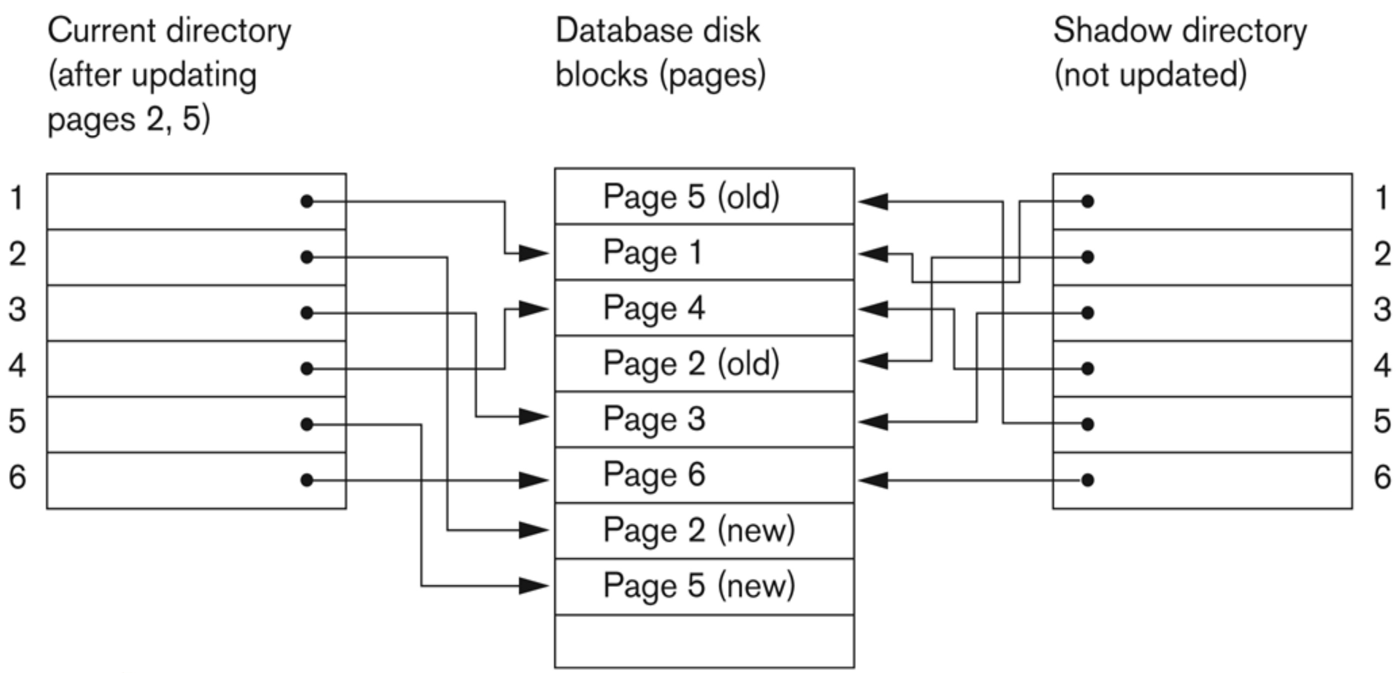


Figure 19.5
An example of
shadow paging.