

Database Reliability

Problem:

How to maintain

atomicity

durability

properties of transactions

10-1

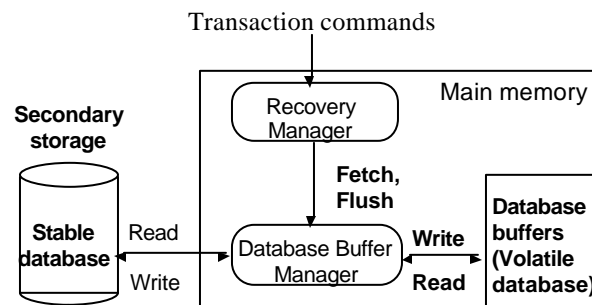
Types of Failures

- Transaction failures
 - Transaction aborts (unilaterally or due to deadlock)
 - Avg. 3% of transactions abort abnormally
- System (site) failures
 - Failure of processor, main memory, power supply, ...
 - Main memory contents are lost, but secondary storage contents are safe
 - Partial vs. total failure
- Media failures
 - Failure of secondary storage devices such that the stored data is lost
 - Head crash/controller failure

10-2

Recovery Management -Architecture

- **Volatile storage**
 - Consists of the main memory of the computer system (RAM).
- **Stable storage**
 - Resilient to failures and loses its contents only in the presence of media failures (e.g., head crashes on disks).
 - Implemented via a combination of hardware (non-volatile storage) and software (stable-write, stable-read, clean-up) components.



10-3

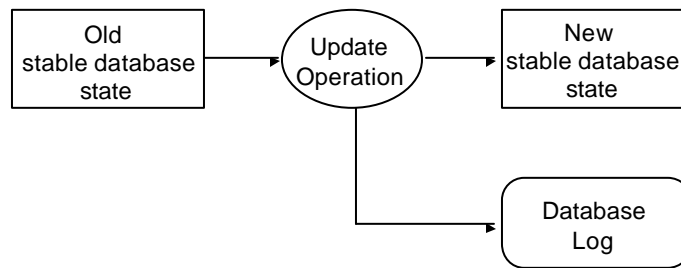
Update Strategies

- **In-place update**
 - Each update causes a change in one or more data values on pages in the database buffers
- **Out-of-place update**
 - Each update causes the new value(s) of data item(s) to be stored separate from the old value(s).
 - **Shadowing**
 - ➔ When an update occurs, don't change the old page, but create a shadow page with the new values and write it into the stable database.
 - ➔ Update the access paths so that subsequent accesses are to the new shadow page.
 - ➔ The old page retained for recovery.

10-4

In-Place Update – Database Log

Every action of a transaction must not only perform the action, but must also write a *log* record to an append-only file.



10-5

Logging

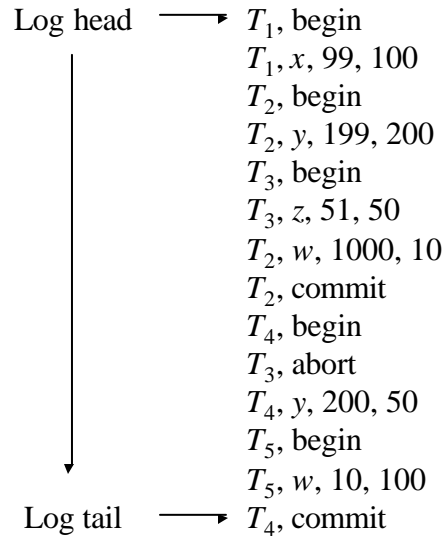
- The log contains information used by the recovery process to restore the consistency of a system.

This information may include

- transaction identifier
- type of operation (action)
- items accessed by the transaction to perform the action
- old value (state) of item (before image)
- new value (state) of item (after image)
- ...

10-6

Log Example

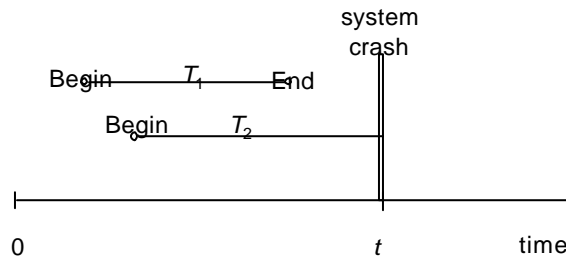


10-7

Why Logging?

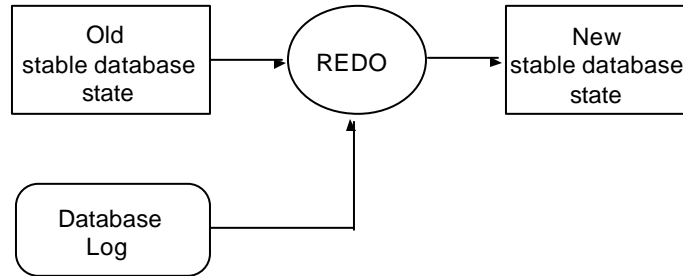
Upon recovery:

- all of T_1 's effects should be reflected in the database (REDO if necessary due to a failure)
- none of T_2 's effects should be reflected in the database (UNDO if necessary)



10-8

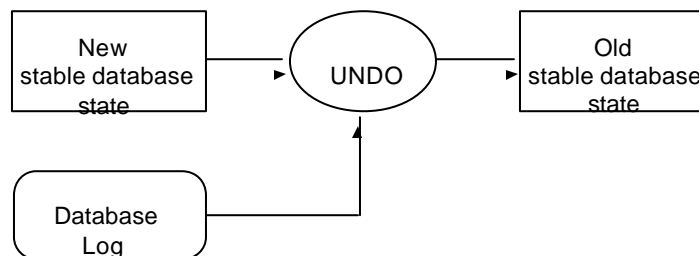
REDO Protocol



- REDO'ing an action means performing it again.
- The REDO operation uses the log information and performs the action that might have already executed or interrupted due to failures.
- The REDO operation generates the new image.

10-9

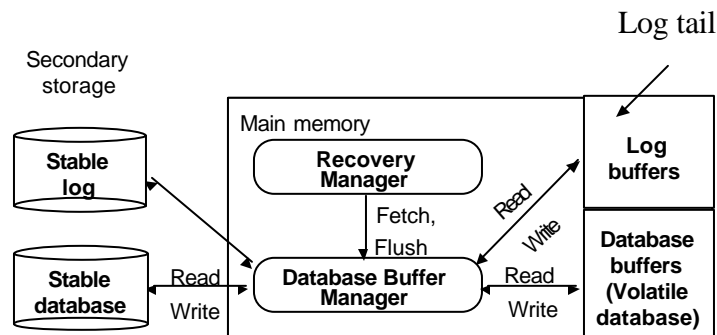
UNDO Protocol



- UNDO'ing an action means to restore the object to its before image.
- The UNDO operation uses the log information and restores the old value of the object.

10-10

Logging Interface



10-11

When to Write Log Records Into Stable Store?

Assume a transaction T updates a page P

■ Fortunate case

- System writes P in stable database
- System updates stable log for this update
- SYSTEM FAILURE OCCURS!... (before T commits)

We can recover (undo) by restoring P to its old state by using the log

■ Unfortunate case

- System writes P in stable database
- SYSTEM FAILURE OCCURS!... (before stable log is updated)

We cannot recover from this failure because there is no log record to restore the old value.

■ Solution: Write-Ahead Log (WAL) protocol

10-12

Write–Ahead Log Protocol

■ Notice:

- If a system crashes before a transaction is committed, then all the operations must be undone. Only need the before images (*undo portion* of the log).
- Once a transaction is committed, some of its actions might have to be redone. Need the after images (*redo portion* of the log).

■ WAL protocol :

- ① Before a stable database is updated, the undo portion of the log should be written to the stable log
- ② When a transaction commits, the redo portion of the log must be written to stable log prior to the updating of the stable database.

10-13

Recovery Manager/Buffer Manager Interaction

- Can the Buffer Manager (BM) decide to write some of the buffer pages being accessed by a transaction into stable storage or does it wait for Recovery Manager (RM) to instruct it?
 - steal/no-steal decision
 - no-steal means RM “pins” (or “fixes”) pages in the buffer
- Does the RM force the BM to write certain buffer pages into stable database at the end of a transaction's execution?
 - force/no-force decision
- Possible execution strategies:
 - steal/no-force
 - steal/force
 - no-steal/no-force
 - no-steal/force

10-14

Steal/No-Force

- **Abort**
 - BM may have written some of the updated pages into stable database
 - RM performs transaction undo (or partial undo)
- **Commit**
 - RM writes a “commit” record into the log.
- **Recover**
 - For those transactions that have both a “begin” and an “commit” record in the log, a partial redo is initiated by RM
 - For those transactions that only have a “begin” record in the log, a global undo is executed by RM

10-15

Steal/Force

- **Abort**
 - BM may have written some of the updated pages into stable database
 - RM performs transaction undo (or partial undo)
- **Commit**
 - RM issues a `flush` command to the buffer manager for all updated pages
 - RM writes a “commit” record into the log.
- **Recover**
 - No need to perform redo
 - Perform global undo

10-16

No-Steal/No-Force

- Abort
 - None of the updated pages have been written into stable database
 - Release the `fixed` pages
- Commit
 - RM writes a “commit” record into the log.
 - RM sends an `unpin` command to the BM for all pages that were previously `pinned`
- Recover
 - Perform partial redo
 - No need to perform global undo

10-17

No-Steal/Force

- Abort
 - None of the updated pages have been written into stable database
 - Release the `fixed` pages
- Commit (the following have to be done atomically)
 - RM issues a `flush` command to the BM for all updated pages
 - RM sends an `unfix` command to the BM for all pages that were previously `fixed`
 - RM writes a “commit” record into the log.
- Recover
 - No need to do anything if page level locking is used
 - May have to undo if finer locking is used

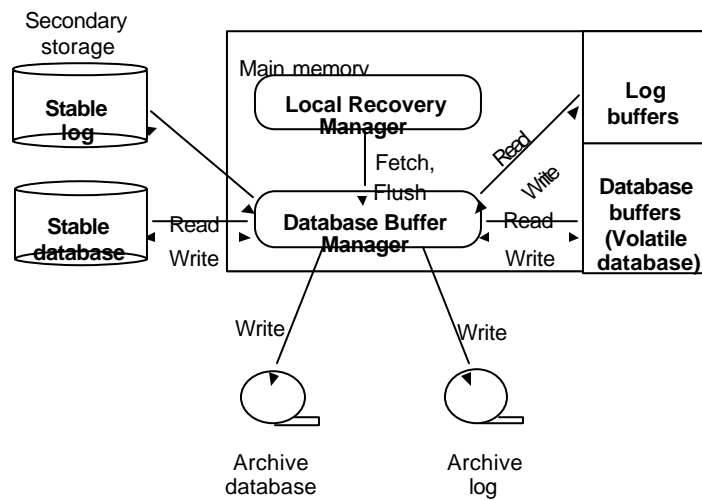
10-18

Checkpoints

- Shortens the amount of log that need to be undone or redone when a failure occurs.
- A checkpoint record contains a list of active transactions.
- Steps:
 - ① Write a begin_checkpoint record into the log
 - ② Collect the checkpoint data into the stable storage
 - ③ Write an end_checkpoint record into the log

10-19

Media Failures – Full Architecture



10-20