The picture can't be displayed.

Recovery System

Database System Concepts, 6th Ed.

Log-Based Recovery

- A **log** is kept on stable storage.
  - The log is a sequence of **log records**, and maintains a record of update activities on the database.
- 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 $<T_i$ **commi**t> is written.
- Two approaches using logs
  - Deferred database modification
  - Immediate database modification

Immediate Database Modification

- The **immediate-modification** scheme allows updates of an uncommitted transaction to be made to the buffer, or the disk itself, before the transaction commits
- Update log record must be written *before* database item is written
  - We assume that the log record is output directly to stable storage
  - (Will see later that how to postpone log record output to some extent)
- Output of updated blocks to stable storage can take place at any time before or after transaction commit
- Order in which blocks are output can be different from the order in which they are written.
- The **deferred-modification** scheme performs updates to buffer/disk only at the time of transaction commit
  - Simplifies some aspects of recovery
  - But has overhead of storing local copy

Transaction Commit

- A transaction is said to have committed when its commit log record is output to stable storage
  - all previous log records of the transaction must have been output already
- Writes performed by a transaction may still be in the buffer when the transaction commits, and may be output later

Immediate Database Modification Example

Log	Write	Output
<T_0 **start**>		
<T_0, A, 1000, 950>		
<T_0, B, 2000, 2050		
	A = 950	
	B = 2050	
<T_0 **commit**>		
<T_1 **start**>		
<T_1, C, 700, 600>		
	C = 600	
<T_1 **commit**>		

B_B , B_C — B_C output before T_1 commits

B_A — B_A output after T_0 commits

- Note: B_X denotes block containing X.

Concurrency Control and Recovery

- With concurrent transactions, all transactions share a single disk buffer and a single log
  - A buffer block can have data items updated by one or more transactions
- We assume that *if a transaction T_i has modified an item, no other transaction can modify the same item until T_i has committed or aborted*
  - i.e. the updates of uncommitted transactions should not be visible to other transactions
    - Otherwise how to perform undo if T1 updates A, then T2 updates A and commits, and finally T1 has to abort?
  - Can be ensured by obtaining exclusive locks on updated items and holding the locks till end of transaction (strict two-phase locking)
- Log records of different transactions may be interspersed in the log.

Undo and Redo Operations

- **Undo** of a log record <T_i, X, V_1, V_2> writes the **old** value V_1 to X
- **Redo** of a log record <T_i, X, V_1, V_2> writes the **new** value V_2 to X
- **Undo and Redo of Transactions**
  - **undo**(T_i) restores the value of all data items updated by T_i to their old values, going backwards from the last log record for T_i
    - each time a data item X is restored to its old value V a special log record <T_i , X, V> is written out
    - when undo of a transaction is complete, a log record <T_i **abort**> is written out.
  - **redo**(T_i) sets the value of all data items updated by T_i to the new values, going forward from the first log record for T_i
    - No logging is done in this case

Undo and Redo on Recovering from Failure

- When recovering after failure:
  - Transaction T_i needs to be undone if the log
    - contains the record <T_i **start**>,
    - but does not contain either the record <T_i **commit**> *or* <T_i **abort**>.
  - Transaction T_i needs to be redone if the log
    - contains the records <T_i **start**>
    - and contains the record <T_i **commit**> *or* <T_i **abort**>
- Note that If transaction T_i was undone earlier and the <T_i **abort**> record written to the log, and then a failure occurs, on recovery from failure T_i is redone
  - **such a redo redoes all the original actions *including the steps that restored old values***
    - Known as **repeating history**
    - Seems wasteful, but simplifies recovery greatly

Immediate DB Modification Recovery Example

Below we show the log as it appears at three instances of time.

(a)	(b)	(c)
<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 commit>	<T_0 commit>
	<T_1 start>	<T_1 start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 commit>

Recovery actions in each case above are:

(a) undo (T_0): B is restored to 2000 and A to 1000, and log records <T_0, B, 2000>, <T_0, A, 1000>, <T_0, **abort**> are written out

(b) redo (T_0) and undo (T_1): *A* and *B* are set to 950 and 2050 and C is restored to 700. Log records <T_1, C, 700>, <T_1, **abort**> are written out.

(c) redo (T_0) and redo (T_1): A and B are set to 950 and 2050 respectively. Then *C* is set to 600