6.4

Recovery System

6.4.1) Recovery Concept

1) Why is recovery needed?

The DBMS must not permit only some of the operations to execute from a transaction, and rest to not executed. This is because the whole transaction is a logical unit of database processing.

If a transaction fails after executing some of its operations but before executing all of them, the operations already executed must be undone and have no lasting effect.

2) Types of failures:

a) A computer failure (system crash):
When a hardware, software, or network error occurs in the computer system during transaction execution.

b) A transaction or system error:
Some operation in the transaction may cause it to fail, such as integer overflow or division by zero.

c) Local errors or exception conditions detected by the transaction:

During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. Eg: Insufficient account balance.

d) Concurrency Control Enforcement:
The concurrency control method may decide to abort a transaction because it violates serializability or causes deadlock.

e) Disk Failure:
Some disk blocks may lose their data because of a read or write malfunction. ~~or because of a head read/write head crash.~~

f) Physical problems & Catastrophies:
A list of problems that includes power or Air Conditioning failure, theft, etc.

3) System log:
Recovery from transaction failures usually means that the database is restored to the most restored to the most recent consistent state just before the time of failure.

To do this, the system must keep information about the changes that were applied to data items by the various transactions.
This information is typically kept in the system log.

4) Deferred Update:
This is a type of techniques do not physically

update the database on disk until after a transaction reaches its commit point; then the updates are recorded in the database.

Commit point : A transaction T reaches its commit point when all its operations are executed successfully and the effect of the transaction operations on the database have been recorded in the log.

Before commit, the updates are recorded on to the log.
After commit, the updates are written to the database on disk.

This technique is also known as the NO-UNDO/REDO algorithm, because if the transaction fails before commiting, the database is not updated & hence we don't need to undo any changes.
However, It may be necessary to REDO committed transactions that were recorded in the log, but weren't yet written on the disk.

Note: Log can contain multiple committed transactions that are not yet written on the database. After some amount, all the transactions are written to the disk (database) together. This is done to reduce cost of writing.

5) Immediate Updates:

Database may be updated by some operations of a transaction before the transaction reaches its commit point.
However, these records must also be recorded in the log, making recovery still possible.

If a transaction fails after recording some changes in the database on disk, the effects of its operations must be undone / rolled - back.
Thus, both undo & redo operations may be required for recovery, that's why it's also called Undo - Redo Algorithm.

6) Caching (Buffering) of disk blocks:

Multiple disk pages that include the data items to be updated are cached into main memory buffers and then updated in memory before written back to disk.

[Note: Page of disk here refers to the specific location of the ~~dis~~ data to be update. Since all data is in the disk, the location of data on disk is required.]

- DBMS cache: A collection of buffers, kept under control of the DBMS for the purpose of storing above buffers

- Directory: Used to keep track of which database items are in the buffer. It ~~is~~ s~~~~

- Dirty Bit: A bit associated with ~~wit~~ each buffer.

  bit is set to 0 when no update has been performed yet.

  bit is set to 1 when an update is performed on cache.

- pin-unpin bit: set to 1 ~~if~~ if a cache cannot be written back to disk.

* Two main strategies can be used to write a modified buffer back to the disk:

i) In-place Updating: Writes the buffer to the same original disk location, overwriting the ~~value~~ old value.

ii) Shadowing: writes an updated buffer at a different disk location, so multiple versions of data items can be maintained.

In general, the old value of the data item being

updated is called the 'before image' (BFIM).

The new value after updating is called the 'after image' (AFIM).

In shadowing, both BFIM and AFIM are stored on disk, hence it's not necessary to keep a log for recovering.

In in-place updating, a log is required to keep track of AFIM and BFIM.

7) Write-Ahead Logging:

In in-place updating the BFIM of the data item is recorded in the appropriate log entry.

The log entry is written to disk before the BFIM is overwritten by the AFIM in the database on disk. This process is generally known as write-ahead logging & is necessary to be able to UNDO the operation if this is required during recovery.

- A Redo-type log entry includes the new value (AFIM) of the item written by the operation since it is needed to redo the effect of operation from log.
- An Undo-type log entry includes the old value (BFIM) of the item since this is needed to undo the effect of the operation from log.

To permit recovery when using the in-place updating method, we use write-ahead logging, since the BFIM is stored in the undo-type record, and can be retrieved in case of system failure.

6.4.2) Log Based Recovery:

1) To be able to recover from failures that affect transactions, the system maintains a log to keep track of all transaction operations that affect the values of database items.

2) The log is a sequential, append only file that is kept on disk, so it is not affected by any type of failure.

3) Main Memory buffers hold the last of the log file, so that log entries are first added to the main memory buffer. This part is called a log-buffer.

4) When the log buffer is filled, it is appended to the log file on disk back.

5) The log file keeps record of the following information:

1. [**start_transaction, *T***]. Indicates that transaction T has started execution.
2. [**write_item, *T*, *X*, *old_value*, *new_value***]. Indicates that transaction T has changed the value of database item X from *old_value* to *new_value*.
3. [**read_item, *T*, *X***]. Indicates that transaction T has read the value of database item X.
4. [**commit, *T***]. Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database. its at commit point
5. [**abort, *T***]. Indicates that transaction T has been aborted.

For recovery from transaction failure, the log is used for undoing or redoing transaction operations individually.

6) Commit Point of a transaction:

A transaction T reaches its commit point when all its operations that access the database have been successfully executed.

After committing, the [commit, T] record is written in the log for T.

After System failure:

- Transactions whose logs have [start transaction, T] but not [commit, T] have to be rolled back / undone.
- Transactions whose logs has [commit, T] will have to be redone using the logs.

7) Using a log buffer saves the overhead of multiple disk writes of the same log buffer. Meaning, if a transaction is changing the value of a data item frequently, then it's not efficient to write them all in the log file that's contained in the disk. Thus log buffers are used, as performing frequent updates on main memory buffer is more efficient.

8) Force writing: Before a transaction reaches its commit point, any portion of the log that has not

been written to the disk must now be written to the disk. This process is called force-writing the log buffer before committing a transaction.

23.6.3.3) Shadow Paging.

1) Shadow paging considers the database to be made up of a number of fixed size disk pages (or blocks) for recovery purposes.

2) A directory with n entries are created where the ith entry points to the ith page on disk. n here is the number of fixed size disk pages / blocks that shadow paging is considering the database to be made of.

3) When a transaction begins execution, the current directory is copied into the 'shadow directory' which is then stored in the disk. Thus, the shadow directory has entries pointing to the disk page which has the initial state of all ~~data~~ items to be operated on.

4) The shadow directory is never updated. Instead, when a write-item operation is performed, a new copy of the modified page is created.

The current directory entry now points to the new disk blocks, but the shadow directory still points to the older disk blocks.

Figure 23.4
An example of shadow paging.