Course Contents

Unit-07:Database Recovery Techniques (3 Hrs.)

- Recovery Concepts; NO-UNDO/REDO Recovery Based on Deferred Update; Recovery Technique Based on Immediate Update; Shadow Paging; Database Backup and Recovery from Catastrophic Failures

Unit 7: Database Recovery Techniques (3Hours) Recovery Concepts *(Recovery outline and categorization of recovery algorithms; Caching (Buffering) of disk blocks; Write-ahead logging, Steal/no-steal, and Force/no-force; Checkpoints and Fuzzy Checkpointing; Transaction rollback and cascading rollback);*	**1Hour**
NO-UNDO/REDO Recovery Based on Deferred Update; Recovery Technique Based on Immediate Update; Shadow Paging; Database Backup and Recovery from Catastrophic Failures;	**2Hours**

Course Contents

Unit-07:Database Recovery Techniques

- Recovery Concepts (Recovery outline and categorization of recovery algorithms;
- Caching (Buffering) of disk blocks;
- Write-ahead logging, Steal/No-steal, and Force/No-force;
- Checkpoints and Fuzzy Check pointing;
- Transaction rollback and cascading rollback;
- NO-UNDO/REDO Recovery Based on Deferred Update;
- Recovery Technique Based on Immediate Update;
- Shadow Paging;
- Database Backup and Recovery from Catastrophic Failures;

Database Recovery

Database Recovery

When a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either

1. all the operations in the transaction are completed successfully and their effect is recorded permanently in the database. or,
2. the transaction has no effect whatsoever on the database or on any other transactions.

Database Recovery

Database Recovery

- Database Recovery is the process by which the database is moved back to a consistent and usable state in case of failure of transaction during transaction execution process.

- So if transaction fails or crashes, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

Database Recovery

Recovery Algorithms:

Recovery algorithms are techniques to ensure database consistency despite failures.

Recovery algorithms have two parts:

1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures. (maintain log)
2. Actions taken after a failure to recover the database contents to a state that ensures data consistency. (recovery technique)

Database Recovery

Failure Classification (What causes a Transaction to fail):

Failures are generally classified as transaction, system and media failures and are of following types:

1) Transaction failure
   a) Logical errors
   b) System errors
2) Disk Failure
3) Concurrency control enforcement
4) Physical problems and catastrophes:

Database Recovery

Failure Classification (What causes a Transaction to fail):

1) Transaction failure

a) Logical errors: i.e. Integer overflow, Division by zero, logical programming etc.

b) System errors: Database system itself terminates an active transaction either not able to execute it or stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

2. Disk Failure :

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

Database Recovery

Failure Classification (What causes a Transaction to fail):

3. Concurrency control enforcement

Example: Violates of serializability may cause to decide abort the transaction.

4. Physical problems and catastrophes:

Endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

Database Recovery

Database recovery techniques/methods are:

Three types of Log-based recovery are:

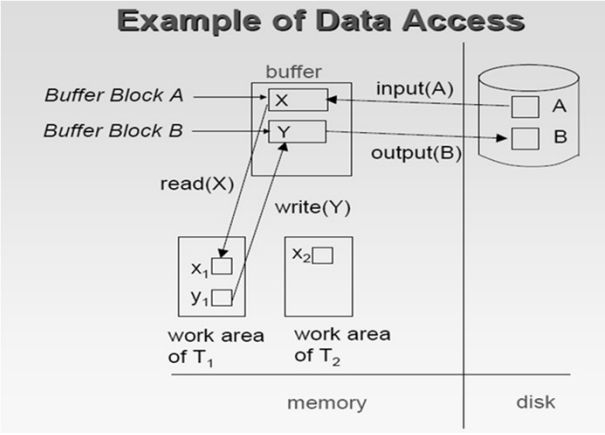
- Deferred database modification − Recovery techniques based on deferred update, also known as the NO-UNDO/REDO technique, where the data on disk is not updated until after a transaction commits. All logs are written on to the stable storage and the database is updated when a transaction commits.
- Immediate database modification − Recovery techniques based on immediate update, where data can be updated on disk during transaction execution; these include the **UNDO/REDO** and **UNDO/NO-REDO** algorithms. Each log follows an actual database modification. That is, the database is modified immediately after every operation.

2. Shadow paging- The shadowing strategy writes an updated buffer at a different disk location, so multiple versions of data items can be maintained. Technique known as shadowing or shadow paging, which can be categorized as a **NO-UNDO/NO-REDO** algorithm.

Database Recovery

Database recovery techniques/methods are:

Three types of Log-based recovery are:



Database Recovery

1. Log-based recovery

- To be able to recover from failures that effect transactions, the system maintains a log to keep track of all transaction operations that affect the values of database items which requires for recovery of data.
- The log is kept on stable storage, so it is not affected by any type of failures except for disk or catastrophic failure.

Because the log contains a record of every write operation that changes the value of some database item, it is possible to undo the effect of these write operations of a transaction T by tracing backward through the log and resetting all items changed by a write operation of T to their old-values.

Database Recovery

1. Log-based recovery

Following are the types of entries that are written in log record (file) and the action each performs.

1. When a transaction enters the system and starts execution, it writes a log about it. <Tn, Start>
2. When the transaction read an item X, it write logs as follows – <Tn, X, V> It reads, Tn read the value of X which is V.
3. When the transaction modifies an item X, it write logs as follows – <Tn, X, V1, V2> It reads, Tn has changed the value of X, from V1 to V2.
4. When the transaction finishes, it logs – <Tn, commit>

Database Recovery

1. Log-based recovery

Some Examples of Log;

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

(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>

Database Recovery

The database can be modified using two approaches:

Deferred database modification − The deferred update techniques do not physically update the database on disk until after a transaction commits; then the updates are recorded in the database. Before reaching commit, all transaction updates are recorded in the local transaction workspace or in the main memory buffers that the DBMS maintains (the DBMS main memory cache). Before commit, the updates are recorded persistently in the log file on disk, and then after commit, the updates are written to the database on disk from the main memory buffers. If a transaction fails before reaching its commit point, it will not have changed the database on disk in any way, so UNDO is not needed. It may be necessary to REDO the effect of the operations of a committed transaction **from the log**, because their effect may not yet have been recorded in the database on disk. Hence, deferred update is also known as the NO-UNDO/REDO algorithm.

Database Recovery

1. Log-based recovery

Immediate database modification − In the immediate update techniques, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations must also be recorded in the log on disk by force-writing before they are applied to the database on disk, making recovery still possible. If a transaction fails after recording some changes in the database on disk but before reaching its commit point, the effect of its operations on the database must be undone; that is, the transaction must be rolled back.

In the general case of immediate update, both undo and redo may be required during recovery. This technique, known as the UNDO/REDO algorithm, requires both operations during recovery and is used most often in practice.

A variation of the algorithm where all updates are required to be recorded in the database on disk before a transaction commits requires undo only, so it is known as the UNDO/NO-REDO algorithm.

Database Recovery

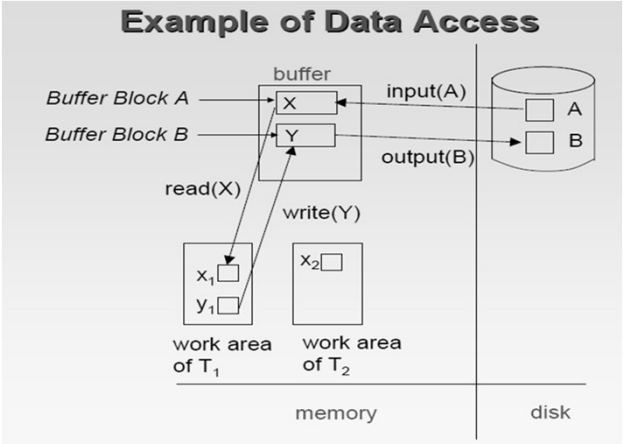
Caching (Buffering) of Disk Blocks:

- The recovery process is often closely intertwined with the buffering of database disk pages in the DBMS main memory cache. Typically, multiple disk pages that include the data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk.
- In general, it is convenient to consider recovery in terms of the database disk pages (blocks). Typically a collection of in-memory buffers, called the DBMS cache, is kept under the control of the DBMS for the purpose of holding these buffers.
- When the DBMS requests action on some item, first it checks the cache directory to determine whether the disk page containing the item is in the DBMS cache. If it is not, the item must be located on disk, and the appropriate disk pages are copied into the cache. It may be necessary to replace (or flush) some of the cache buffers to make space available for the new item.

Database Recovery

Database Buffering (Buffer Management):

Databases are maintained in non-volatile memory (HDD) where HDD are managed to store the database as sector, buffer. Sector normally of size 512 bytes and one block is of many sectors. Data from HDD is transferred to memory in block by block for processing the data.



Database Recovery

Database Buffering (Buffer Management):

- We need to use disk storage for the database, and to transfer blocks of data between Main Memory(MM) and disk.
- We also want to minimize the number of such transfers, as they are time-consuming.
- One way is to keep as many blocks as possible in MM.
- Usually, we cannot keep all blocks in MM, so we need to manage the allocation of available MM space.
- The buffer is the part of MM available for storage of copies of disk blocks.
- The subsystem responsible for the allocation of buffer space is called the buffer manager.
- The buffer manager handles all requests for blocks of the database.
- If the block is already in MM, the address in MM is given to the requestor.
- If not, the buffer manager must read the block in from disk (possibly displacing some other block if the buffer is full) and then pass the address in MM to the requestor.

Database Recovery

Caching (Buffering) of Disk Blocks:

Dirty bit- The entries in the DBMS cache directory hold additional information relevant to buffer management. Associated with each buffer in the cache is a dirty bit, which can be included in the directory entry to indicate whether or not the buffer has been modified. When a page is first read from the database disk into a cache buffer, a new entry is inserted in the cache directory with the new disk page address, and the dirty bit is set to 0 (zero). As soon as the buffer is modified, the dirty bit for the corresponding directory entry is set to 1 (one).

Additional information, such as the transaction id(s) of the transaction(s) that modified the buffer, are also kept in the directory.

When the buffer contents are replaced (flushed) from the cache, the contents must first be written back to the corresponding disk page only if its dirty bit is 1.
Note: Dirty bit for indication of whether buffer contents are changed or not.

Database Recovery

Caching (Buffering) of Disk Blocks:

pin-unpin - a page in the cache is pinned (bit value 1 (one)) if it cannot be written back to disk as yet. For example, the recovery protocol may restrict certain buffer pages from being written back to the disk until the transactions that changed this buffer have committed.

Note: pin-unpin bit for indication of whether buffer contents are written back to disk yet or not.

Database Recovery

Caching (Buffering) of Disk Blocks:

Two main strategies can be employed when flushing a modified buffer back to disk.

- The first strategy, known as in-place updating, writes the buffer to the same original disk location, thus overwriting the old value of any changed data items on disk. Hence, a single copy of each database disk block is maintained.
- The second strategy, known as shadowing, writes an updated buffer at a different disk location, so multiple versions of data items can be maintained, but this approach is not typically used in practice.
- In general, the old value of the data item before updating is called the before image (BFIM), and the new value after updating is called the after image (AFIM).
- If shadowing is used, both the BFIM and the AFIM can be kept on disk; hence, it is not strictly necessary to maintain a log for recovering.

Database Recovery

Write-ahead logging, Steal/no steal, and Force/no-force:

Write-ahead logging − DBMS cache holds the cached database disk blocks in main memory buffers, which include not only data blocks, but also index blocks and log blocks from the disk.

When a log record is written, it is stored in the current log buffer in the DBMS cache. When an update to a data block stored in the DBMS cache is made, an associated log record is written to the last log buffer in the DBMS cache.

With the write-ahead logging approach, the log buffers (blocks) that contain the associated log records for a particular data block update must first be written to disk before the data block itself can be written back to disk from its main memory buffer.

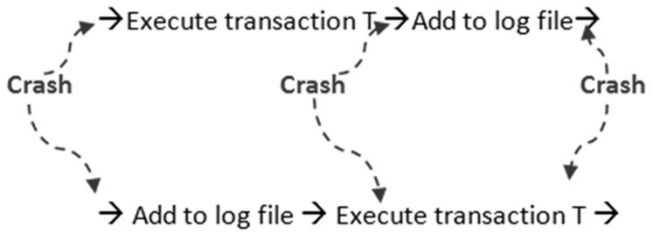
So, **Write-ahead logging** is writing the log entry to disk before the BFIM is overwritten with the AFIM in the database on disk.

Database Recovery

Write-ahead logging, Steal/no steal, and Force/no-force:

Write-ahead logging − The recovery mechanism must ensure that the BFIM of the data item is recorded in the appropriate log entry and that the log entry is flushed to disk before the BFIM is overwritten with the AFIM in the database on disk.

This process is generally known as **write-ahead logging** and is necessary so we can UNDO the operation if this is required during recovery.



Database Recovery

Write-ahead logging, Steal/no steal, and Force/no-force:

Write-ahead logging −

Briefly, WAL's central concept is that changes to data files must be written only after those changes have been logged, that is, after log records describing the changes have been flushed to permanent storage.

If we follow this procedure, we do not need to flush data pages to disk on every transaction commit, because we know that in the event of a crash we will be able to recover the database using the log: any changes that have not been applied to the data pages can be redone from the log records.

Database Recovery

Write-ahead logging, Steal/no steal, and Force/no-force:

For write-ahead logging, we need to distinguish between two types of log entry information included for a write command: the information needed for UNDO and the information needed for REDO.

- A REDO-type log entry includes the new value (AFIM) of the item written by the operation since this is needed to redo the effect of the operation from the log (by setting the item value in the database on disk to its AFIM).
- The UNDO-type log entries include the old value (BFIM) of the item since this is needed to undo the effect of the operation from the log (by setting the item value in the database back to its BFIM).
- In an UNDO/REDO algorithm, both BFIM and AFIM are recorded into a single log entry.

Database Recovery

Steal/no steal, and Force/no-force:

Standard DBMS recovery terminology includes the terms steal/no-steal and force/no-force, which specify the **rules that govern when a page from the database cache can be written to disk**:

1. If a cache buffer page updated by a transaction cannot be written to disk before the transaction commits, the recovery method is called a **no-steal** approach. The pin-unpin bit will be set to 1 (pin) to indicate that a cache buffer cannot be written back to disk. On the other hand, if the recovery protocol allows writing an updated buffer before the transaction commits, it is called steal. The no-steal rule means that UNDO will never be needed during recovery, since a committed transaction will not have any of its updates on disk before it commits.
2. If all pages updated by a transaction are immediately written to disk before the transaction commits, the recovery approach is called a **force approach**. Otherwise, it is called **no-force**. The force rule means that REDO will never be needed during recovery, since any committed transaction will have all its updates on disk before it is committed.

Database Recovery

Checkpoint

When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

- Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all.
- Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Database Recovery

Checkpoint

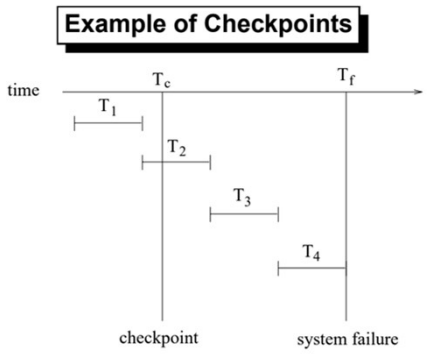
Checkpoint is a another type of entry in the log. A [checkpoint, list of active transactions] record is written into the log periodically at that point when the system writes out to the database on disk all DBMS buffers that have been modified.

As a consequence of this, all transactions that have their [commit, T] entries in the log before a [checkpoint] entry do not need to have their WRITE operations redone in case of a system crash, since all their updates will be recorded in the database on disk during check pointing.

As part of check pointing, the list of transaction ids for active transactions at the time of the checkpoint is included in the checkpoint record, so that these transactions can be easily identified during recovery.

Database Recovery

Checkpoint



- T_1 can be ignored (updates already output to disk due to checkpoint)
- T_2 and T_3 redone
- T_4 undone

Database Recovery

Checkpoint:

Taking a checkpoint consists of the following actions:

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

As a consequence of step 2, a checkpoint record in the log may also include additional information, such as a list of active transaction ids, and the locations (addresses) of the first and most recent (last) records in the log for each active transaction. This can facilitate undoing transaction operations in the event that a transaction must be rolled back.

Database Recovery

Fuzzy checkpointing:

- The time needed to force-write all modified memory buffers may delay transaction processing because of step 1, which is not acceptable in practice. To overcome this, it is common to use a technique called fuzzy checkpointing. In this technique, the system can resume transaction processing after a [begin_checkpoint] record is written to the log without having to wait for step 2 to finish.
- When step 2 is completed, an [end_checkpoint, …] record is written in the log with the relevant information collected during checkpointing. However, until step 2 is completed, the previous checkpoint record should remain valid. To accomplish this, the system maintains a file on disk that contains a pointer to the valid checkpoint, which continues to point to the previous checkpoint record in the log.
- Once step 2 is concluded, that pointer is changed to point to the new checkpoint in the log.

Database Recovery

Transaction Rollback and Cascading Rollback:

Transaction Rollback- If a transaction fails for whatever reason after updating the database, but before the transaction commits, it may be necessary to roll back the transaction. If any data item values have been changed by the transaction and written to the database on disk, they must be restored to their previous values (BFIMs). The undo-type log entries are used to restore the old values of data items that must be rolled back.

Cascading Rollback- If a transaction T is rolled back, any transaction S that has, in the interim, read the value of some data item X written by T must also be rolled back. Similarly, once S is rolled back, any transaction R that has read the value of some data item Y written by S must also be rolled back; and so on. This phenomenon is called cascading rollback.

Database Recovery

NO-UNDO/REDO Recovery Based on Deferred Update:

- The idea behind deferred update is to defer or postpone any actual updates to the database on disk until the transaction completes its execution successfully and reaches its commit point.
- During transaction execution, the updates are recorded only in the log and in the cache buffers. 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 operations because the transaction has not affected the database on disk in any way.
- Therefore, only REDO type log entries are needed in the log, which include the new value (AFIM) of the item written by a write operation. The UNDO-type log entries are not needed since no undoing of operations will be required during recovery.

Database Recovery

NO-UNDO/REDO Recovery Based on Deferred Update:

- Because the database is never updated on disk until after the transaction commits, there is never a need to UNDO any operations. REDO is needed in case the system fails after a transaction commits but before all its changes are recorded in the database on disk. In this case, the transaction operations are redone from the log entries during recovery.

We can state a typical deferred update protocol as follows:

1. A transaction cannot change the database on disk until it reaches its commit point; hence all buffers that have been changed by the transaction must be pinned until the transaction commits (this corresponds to a no-steal policy).
2. A transaction does not reach its commit point until all its REDO-type log entries are recorded in the log and the log buffer is force-written to disk.

Database Recovery

Recovery Techniques Based on Immediate Update:

In these techniques, when a transaction issues an update command, the database on disk can be updated immediately, without any need to wait for the transaction to reach its commit point.

Provisions must be made for undoing the effect of update operations that have been applied to the database by a failed transaction. This is accomplished by rolling back the transaction and undoing the effect of the transaction's write_item operations.

Therefore, the UNDO-type log entries, which include the old value (BFIM) of the item, must be stored in the log. Because UNDO can be needed during recovery, these methods follow a steal strategy for deciding when updated main memory buffers can be written back to disk.

Database Recovery

Recovery Techniques Based on Immediate Update:

Theoretically, we can distinguish two main categories of immediate update algorithms.

1. If the recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is never a need to REDO any operations of committed transactions. This is called the UNDO/NO-REDO recovery algorithm. In this method, all updates by a transaction must be recorded on disk before the transaction commits, so that REDO is never needed. Hence, this method must utilize the steal/force strategy for deciding when updated main memory buffers are written back to disk.
2. If the transaction is allowed to commit before all its changes are written to the database, we have the most general case, known as the UNDO/REDO recovery algorithm. In this case, the steal/no-force strategy is applied.

Database Recovery

Shadow Paging:

- Shadow paging considers the database to be made up of a number of fixed size disk pages (or disk blocks) say, n for recovery purposes. A directory with n entries is constructed, where the i^{th} entry points to the i^{th} database page on disk.
- When a transaction begins executing, the current directory whose entries point to the most recent or current database pages on disk is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction.
- During transaction execution, the shadow directory is never modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten. Instead, the new page is written elsewhere on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block.

Database Recovery

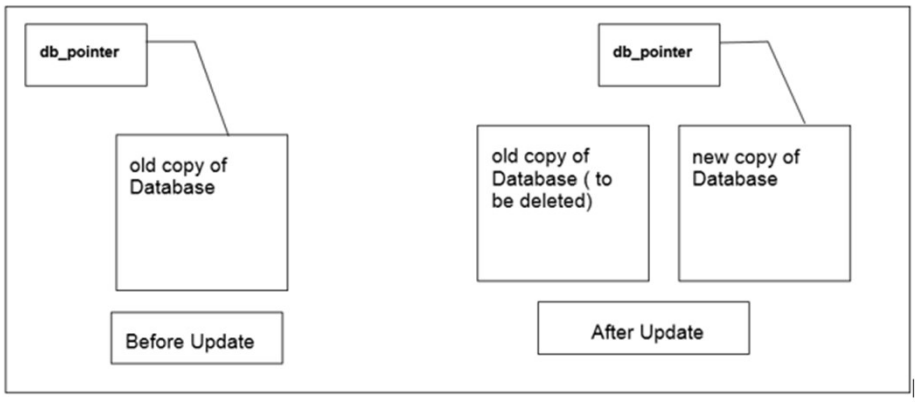
Shadow Paging:

- For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory and the new version by the current directory.
- To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory. The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory.
- The database thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded.
- Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NO-UNDO/NO-REDO technique for recovery.

Database Recovery

3. Shadow Paging:

The shadow database scheme:



- Assumes disks to not fail
- Useful for text editors, but extremely inefficient for large databases
- Executing a single transaction requires copying the entire database.

Database Recovery

Database Backup and Recovery from Catastrophic Failures:

- So far, all the techniques we have discussed apply to noncatastrophic failures. A key assumption has been that the system log is maintained on the disk and is not lost as a result of the failure. Similarly, the shadow directory must be stored on disk to allow recovery when shadow paging is used. The recovery techniques we have dis-cussed use the entries in the system log or the shadow directory to recover from failure by bringing the database back to a consistent state.
- The recovery manager of a DBMS must also be equipped to handle more catastrophic failures such as disk crashes. The main technique used to handle such crashes is a database backup, in which the whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes or other large capacity offline storage devices. In case of a catastrophic system failure, the latest backup copy can be reloaded from the tape to the disk, and the system can be restarted.

Database Recovery

Database Backup and Recovery from Catastrophic Failures:

- To avoid losing all the effects of transactions that have been executed since the last backup, it is customary to back up the system log at more frequent intervals than full database backup by periodically copying it to magnetic tape.
- The system log is usually substantially smaller than the database itself and hence can be backed up more frequently. Therefore, users do not lose all transactions they have performed since the last database backup. All committed transactions recorded in the portion of the system log that has been backed up to tape can have their effect on the data-base redone. A new log is started after each database backup. Hence, to recover from disk failure, the database is first recreated on disk from its latest backup copy on tape. Following that, the effects of all the committed transactions whose operations have been recorded in the backed-up copies of the system log are reconstructed.