**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**theory Assignment # 9**

**Submitted by:**

PRANESH DHUNJU SHRESTHA  
013BSCCSIT026

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav**  Lecturer  St. Xavier’s College |  |

**Date of Submission:** October 1, 2015

## DATABASE RECOVERY

## 1. Purpose to Database Recovery

A major responsibility of the database administrator is to prepare for the possibility of hardware, software, network, process, or system failure. If such a failure affects the operation of a database system, you must usually recover the database and return to normal operation as quickly as possible. Recovery should protect the database and associated users from unnecessary problems and avoid or reduce the possibility of having to duplicate work manually.

Recovery processes vary depending on the type of failure that occurred, the structures affected, and the type of recovery that you perform. If no files are lost or damaged, recovery may amount to no more than restarting an instance. If data has been lost, recovery requires additional steps.

**2. Types of Failure**

To see where the problem has occurred, we generalize a failure into various categories, as follows −

* + - **Transaction failure**

A transaction has to abort when it fails to execute or when it reaches a point from where it can’t go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be −

* **Logical errors** − Where a transaction cannot complete because it has some code error or any internal error condition.
* **System errors** − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.
* **System Crash**

There are problems − external to the system − that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

Examples may include operating system errors.

* **Disk Failure**

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

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. [1]

## 3. Storage Hierarchy

We have already described the storage system. In brief, the storage structure can be divided into two categories :

* **Volatile storage** − As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
* **Non-volatile storage** − These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

# 4. Buffer Management

* + We need to use disk storage for the database, and to transfer blocks of data between 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.

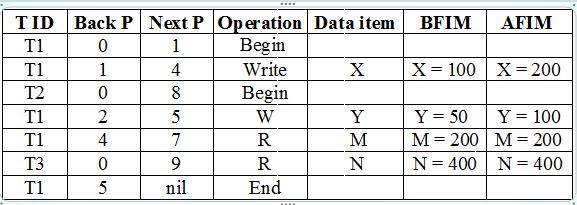
1. The buffer manager must use some sophisticated techniques in order to provide good service:
   * **Replacement Strategy** - When there is no room left in the buffer, some block must be removed to make way for the new one. Typical operating system memory management schemes use a ``least recently used'' (**LRU**) method. (Simply remove the block least recently referenced.) This can be improved upon for database applications.
   * **Pinned Blocks** - For the database to be able to recover from crashes, we need to restrict times when a block maybe written back to disk. A block not allowed to be written is said to be **pinned**. Many operating systems do not provide support for pinned blocks, and such a feature is essential if a database is to be ``crash resistant''.
   * **Forced Output of Blocks** - Sometimes it is necessary to write a block back to disk even though its buffer space is not needed. (Called the **forced output** of a block.) This is due to the fact that MM contents (and thus the buffer) are lost in a crash, while disk data usually survives.
2. **Replacement Strategy:** Goal is minimization of accesses to disk. Generally it is hard to predict which blocks will be referenced. So operating systems use the history of past references as a guide to prediction.
   * **General Assumption:** Blocks referenced recently are likely to be used again.
   * **Therefore:** if we need space, throw out the least recently referenced block. (LRU replacement scheme).

# 5.The Transaction Log

Every database has a transaction log that records all transactions and the database modifications made by each transaction. The transaction log must be truncated on a regular basis to keep it from filling up. However, some factors can delay log truncation, so monitoring log size is important. Some operations can be minimally logged to reduce their impact on transaction log size.

The transaction log is a critical component of the database and, if there is a system failure, the transaction log might be required to bring your database back to a consistent state. The transaction log should never be deleted or moved unless you fully understand the ramifications of doing this.

A DBMS uses a transaction log to keep track of all transactions that update the database. The information stored in this log is used by the DBMS for a recovery requirement triggered by a ROLLBACK statement, a program’s abnormal termination, or a system failure such as a network discrepancy or a disk crash. Some RDBMSs use the transaction log to recover a database forward to a currently consistent state. After a server failure, for example, Oracle automatically rolls back uncommitted transactions and rolls forward transactions that were committed but not yet written to the physical database.



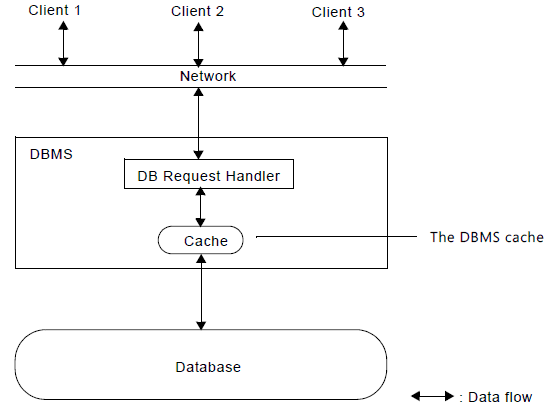
**6. Data Update**

* + **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.
  + **In-place update**: The disk version of the data item is overwritten by the cache version.

# 7. Data caching

The Database Management System (DBMS) is a memory buffer which stores copies of portions of the database that the DBMS is currently using. Reading from memory is much faster than reading from the disk. The DBMS therefore returns a record more quickly if it is already stored in cache. As long as the required data is stored in cache, the data is immediately available. When the required data is not stored in cache, it must be copied from the disk and then stored in cache.

For example, three users send requests to the DBMS. When user 2 sends a request to read data from the database, the request handler determines whether the desired data can be fetched directly from the cache or whether it must be fetched from a disk.



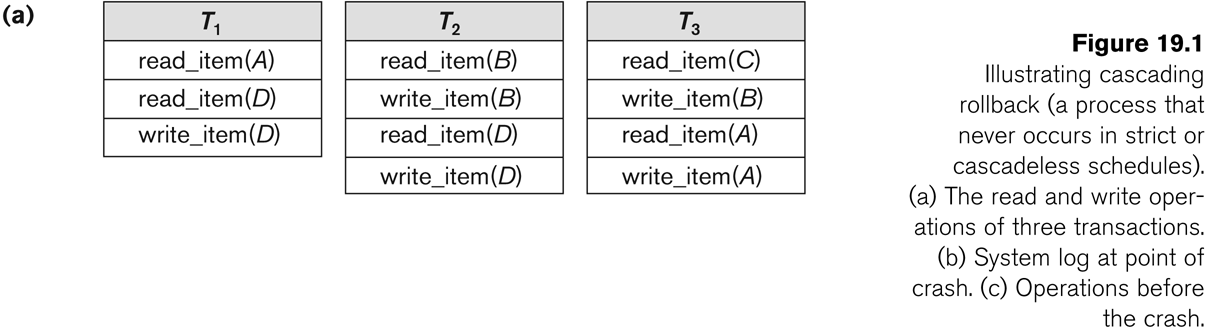
The DBMS cache always contains the most recently used data. The cache is continually updated with the relevant data from the database.

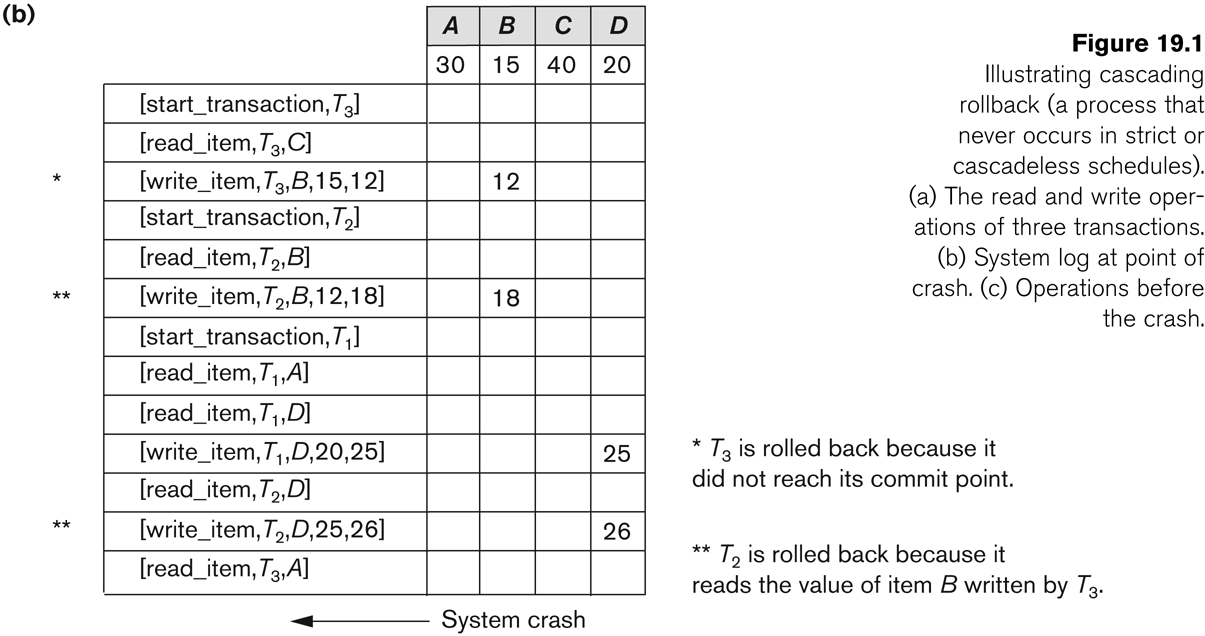
The size of the cache greatly affects performance. When you set the size of the cache, you must remember two simple rules:

* The more memory you assign to the cache, the more efficient it becomes. (Of course, there is no reason to assign more memory to the cache than the total size of your database.)
* The size of the cache must not exceed the amount of physical memory available on your system. This is because the operating system may swap the cache memory in and out of the disk. This will considerably slow down overall performance.

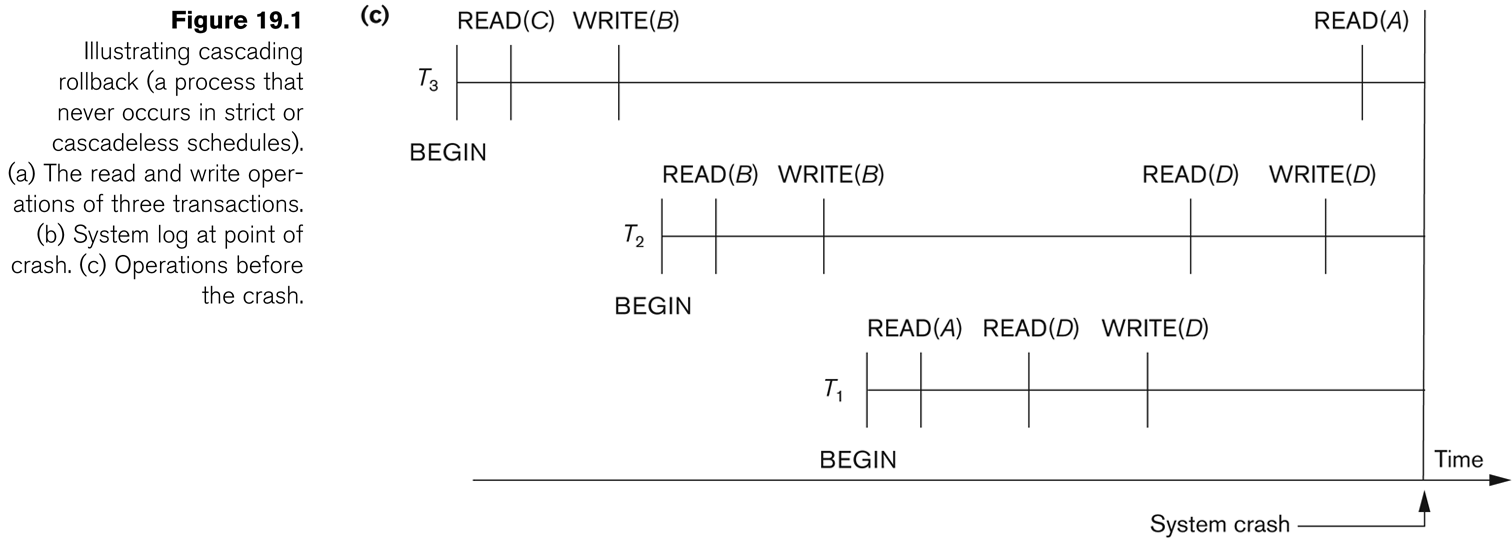
**8. Transaction Roll-back (Undo) and Roll-Forward (Redo)**

* + 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.





**Roll-back**: One execution of T1, T2 and T3 as recorded in the log.

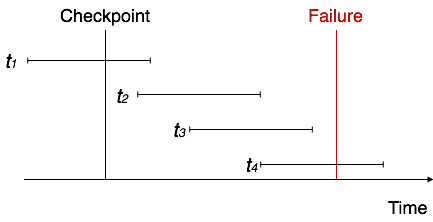


**9. Checkpoint**

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.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −

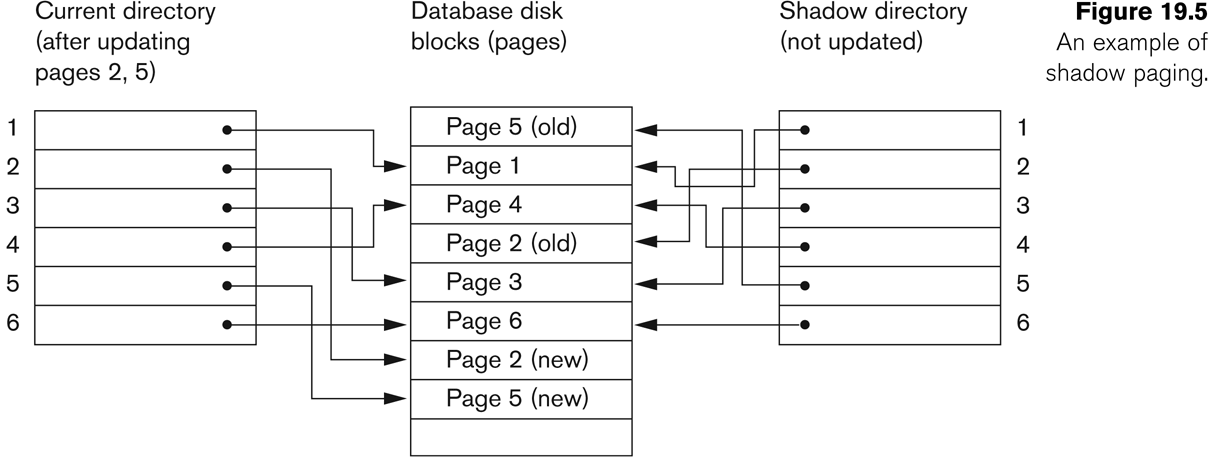


* The recovery system reads the logs backwards from the end to the last checkpoint.
* It maintains two lists, an undo-list and a redo-list.
* If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
* If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

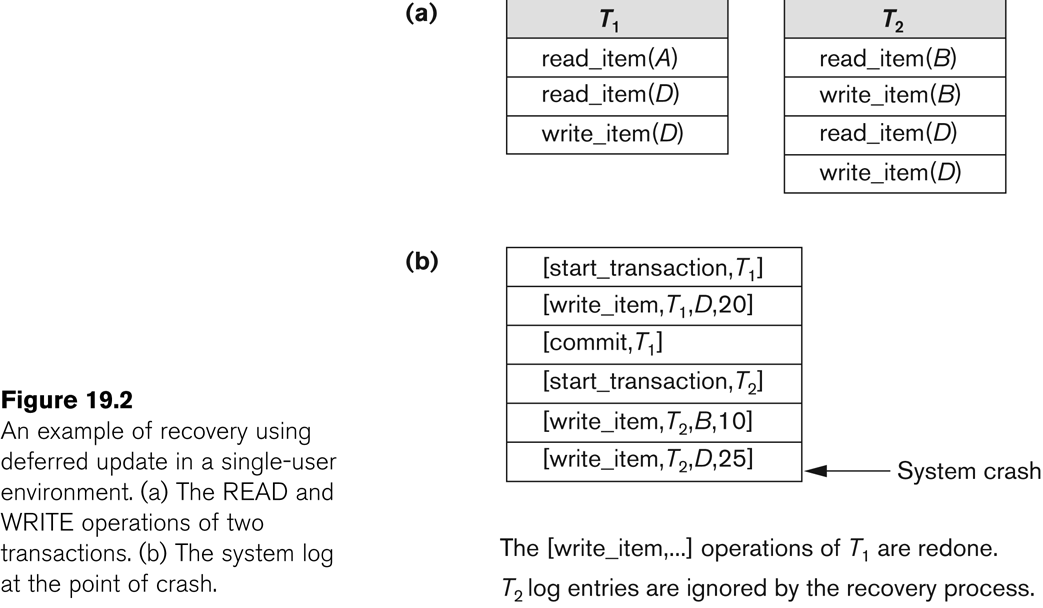
**Shadow Paging**

* To manage access of data items by concurrent transactions two directories (current and shadow) are used.
  + The directory arrangement is illustrated below. Here a page is a data item.



**10. Recovery Schemes**

* Deferred Update (No Undo/Redo)
  + The data update goes as follows:
  + A set of transactions records their updates in the log.
  + At commit point under WAL scheme these updates are saved on database disk.
  + After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.
* Deferred Update in a single-user system  
  There is no concurrent data sharing in a single user system. The data update goes as follows:
  + A set of transactions records their updates in the log.
  + At commit point under WAL scheme these updates are saved on database disk.
* After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.



* **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.

**11 . Failure with Loss of Non-volatile Storage**

* So far we assumed no loss of non-volatile storage.
* Technique similar to checkpointing used to deal with loss of nonvolatile storage .
* Periodically dump the entire content of the database to stable storage .
* No transaction may be active during the dump procedure; a procedure similar to checkpointing must take place.
* Output all log records currently residing in main memory onto stable storage.
* Output all buffer blocks onto the disk.
* Copy the contents of the database to stable storage.
* Output a record <dump> to log on stable storage.

**12. Recovery in multidatabase system**

* A multidatabase system is a special distributed database system where one node may be running relational database system under UNIX, another may be running object-oriented system under Windows and so on.
* A transaction may run in a distributed fashion at multiple nodes.
* In this execution scenario the transaction commits only when all these multiple nodes agree to commit individually the part of the transaction they were executing.
* This commit scheme is referred to as “**two-phase commit**” (**2PC**).
  + If any one of these nodes fails or cannot commit the part of the transaction, then the transaction is aborted.
* Each node recovers the transaction under its own recovery protocol.

**Reference:**

[1] http://www.tutorialspoint.com/dbms/dbms\_data\_recovery.htm

[2] http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chap7/node11.html

[3] http://codex.cs.yale.edu/avi/db-book/db5/slide-dir/ch17.pdf

[4] https://msdn.microsoft.com/en-us/library/dd355169.aspx

[5] https://www.cs.purdue.edu/homes/ake/cs348/Chapter19.ppt