ST.XAVIER’S COLLEGE

**Maitighar, Kathmandu**



**DBMS LAB ASSIGNMENT**

**Submitted By:**

**Narayan Panthi**

013BSCCSIT024

**Submitted To:**

**Er. Sanjay Kumar Yadav** ………..…………….

**Lecturer**

**Department of Computer Science**

**Date of Submission:- *2015 -*** *\_\_****\_-*** *\_\_\_*

# Database Recovery

Data recovery is the process of restoring data that has been lost, accidentally deleted, corrupted or made inaccessible for any reason.

Reconstructing the contents of all or part of a database from a backup typically involves two phases: retrieving a copy of the data file from a backup, and reapplying changes to the file since the backup from the archived and online redo logs, to bring the database to a desired SCN since the backup (usually, the present).

# The Storage Hierarchy

Databases are stored in file formats, which contain records. At physical level, the actual data is stored in electromagnetic format on some device. These storage devices can be broadly categorized into three types −



* **Primary Storage** − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.
* **Secondary Storage** − Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.
* **Tertiary Storage** − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

The storage structure can be divided into two categories −

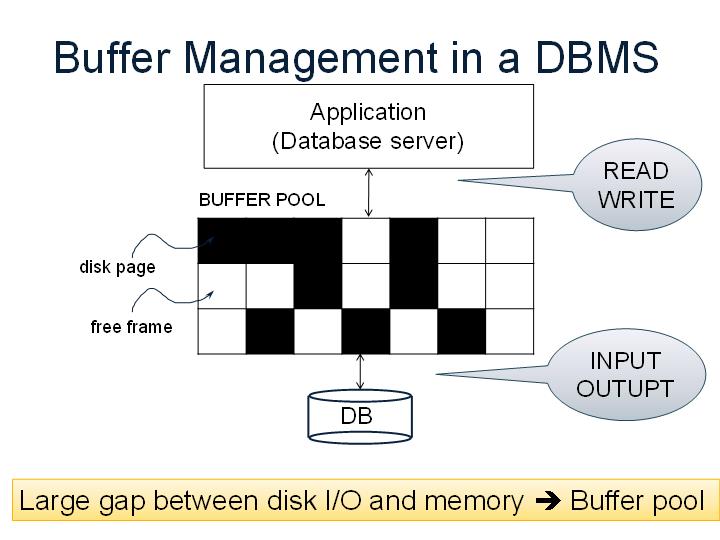
* **Volatile storage** − As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but 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
  + **The Problem of Database Recovery**
    - To restore the database to a **state** that is known to be **correct** after some failure.
  + **Principle of Recovery:** 
    - Backup is necessary
* **Basic approach**
* Dump database periodically.
  + - Write a log record for every change. e.g. E#, old\_value, new\_value, …
    - If a failure occurs:

CASE1 : DB is damaged   
 ==> archive copy + redo log = current DB.

CASE2 : DB is not damaged but contents unreliable   
 ==> undo some log.

# Purpose of Database Recovery

* + To bring the database into the last consistent state, which existed prior to the failure.
  + To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).
  + Example:
    - If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

**Buffer Management**

**When a Page is Requested ...**

* Buffer pool information table contains:

<frame#, pageid, pin\_count, dirty>

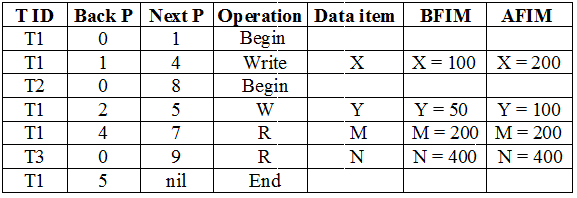
* **If requested page is not in pool:**
  + Choose a frame for replacement.
  + Only “un-pinned” pages are candidates!
  + If frame is “dirty”, write it to disk
  + Read requested page into chosen frame
  + Pin the page and return its address.
  + If requests can be predicted (e.g., sequential scans), pages can be pre-fetched several pages at a time.

# Types of Failure

* The database may become unavailable for use due to
  + - **Transaction failure**: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
    - **System failure**: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
    - **Media failure**: Disk head crash, power disruption, etc.

# Transaction Log

* + For recovery from any type of failure data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required.
  + These values and other information is stored in a sequential file called Transaction log. A sample log is given below. Back P and Next P point to the previous and next log records of the same transaction.



# 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.

# Data Caching

* + Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.
  + The flushing is controlled by **Modified** and **Pin-Unpin** bits.
    - **Pin-Unpin**: Instructs the operating system not to flush the data item.
    - **Modified**: Indicates the AFIM of the data item.

# 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.

# Checkpointing

* + Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:
    1. Suspend execution of transactions temporarily.
    2. Force write modified buffer data to disk.
    3. Write a [checkpoint] record to the log, save the log to disk.
    4. Resume normal transaction execution.

During recovery redo or undo is required to transactions appearing after record.

# Shadow Paging

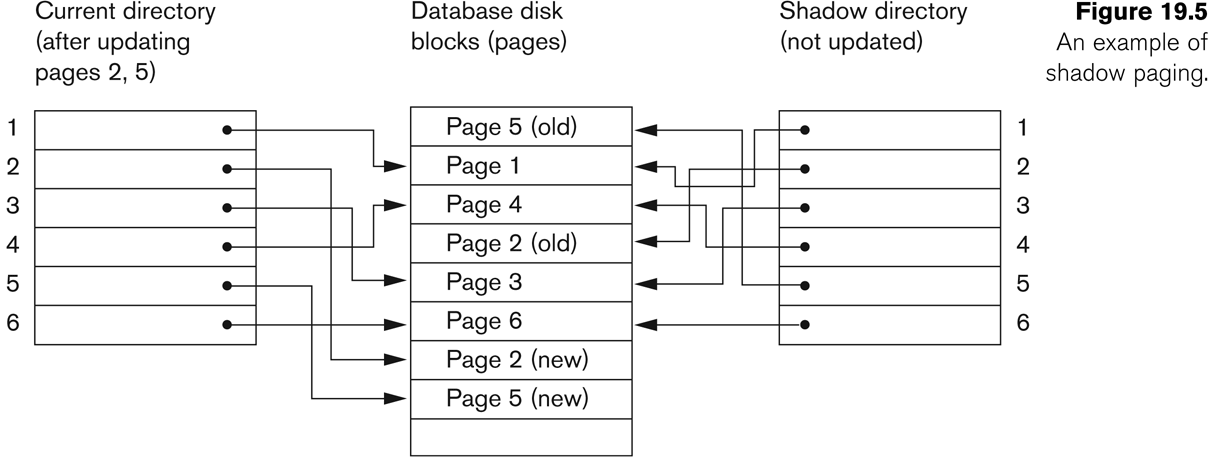
* The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.



X and Y: Shadow copies of data items

X' and Y': Current copies of data item.

* 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.



# Recovery Scheme

* 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.

# (WAL: Write Ahead Logging Protocol)

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.

# Failure with loss of Non-volatile Storage (General Concepts)

A volatile storage like RAM stores all the active logs, disk buffers, and related data. In addition, it stores all the transactions that are being currently executed. What happens if such a volatile storage crashes abruptly? It would obviously take away all the logs and active copies of the database. It makes recovery almost impossible, as everything that is required to recover the data is lost.

Following techniques may be adopted in case of loss of volatile storage −

* We can have checkpoints at multiple stages so as to save the contents of the database periodically.
* A state of active database in the volatile memory can be periodically dumped onto a stable storage, which may also contain logs and active transactions and buffer blocks.
* <dump> can be marked on a log file, whenever the database contents are dumped from a non-volatile memory to a stable one.

# Recovery in Multi database System

To maintain the atomicity of a multi database transaction, it is necessary to have a two-level recovery mechanism. A global recovery manager, or coordinator, is needed to maintain information needed for recovery, in addition to the local recovery managers and the information they maintain (log, tables).

The coordinator usually follows a protocol called the **two-phase commit protocol,** whose two phases can be stated as follows:

• **Phase 1:** When all participating databases signal the coordinator that the part of the multidatabase transaction involving each has concluded, the coordinator sends a message "prepare for commit" to each participant to get ready for committing the transaction.

• **Phase 2:** If *all* participating databases reply "OK," and the coordinator’s vote is also "OK," the transaction is successful, and the coordinator sends a "commit" signal for the transaction to the participating databases. Because all the local effects of the transaction and information needed for local recovery have been recorded in the logs of the participating databases, recovery from failure is now possible. Each participating database completes transaction commit by writing a [commit] entry for the transaction in the log and permanently updating the database if needed.