**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Lab Assignment #9**

**Submitted by:**

Arun Sanjel  
013BSCCSIT010

**Submitted to:**

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

# Date of Submission: Friday, October 02, 2015Database Recovery

## Purpose of Data 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.

## Types of failure

Failures may be

|  |  |
| --- | --- |
| Transaction | Caused by errors within the transaction processes. |
| System | Caused by failure of network or operating system or physical threats to the system as a whole. |
| Media | Failure of hard disk, out of memory errors, out of disk space errors. |

## Reasons for Failure

Failure may be caused by a number of things.

|  |  |
| --- | --- |
| A System Crash | A hardware, software or network error causes the transaction to fail. |
| Transaction or System error | Some operation in the transaction may cause the failure or the user may interrupt the transaction. |
| Local Errors or Exceptions | Conditions occur during the transaction that results in transaction cancellation. |
| Concurrency Control Enforcement | Several transactions may be in deadlock so the transaction may be aborted to be restarted later. |
| Disk Failure | Read Write error on the physical disk. |
| Physical Problems | This can be any range of physical problems, such as power failure, mounting wrong disk or tape by operator, wiring problems etc |
| Catastrophe Situations | Large scale threats to the system and the data for example fire, cyclone, security breaches etc. |

## The Storage Hierarchy

Data are the principal resources of an organization. Data stored in computer systems form a hierarchy extending from a single bit to a database, the major record-keeping entity of a firm. Each higher rung of this hierarchy is organized from the components below it. Data are logically organized into:

**Bit (Character)** - a bit is the smallest unit of data representation (value of a bit may be a 0 or 1). Eight bits make a byte which can represent a character or a special symbol in a character code.

**Field** - a field consists of a grouping of characters. A data field represents an attribute (a characteristic or quality) of some entity (object, person, place, or event).

**Record** - a record represents a collection of attributes that describe a real-world entity. A record consists of fields, with each field describing an attribute of the entity.

**File** - a group of related records. Files are frequently classified by the application for which they are primarily used (employee file). A primary key in a file is the field (or fields) whose value identifies a record among others in a data file.

**Database** - is an integrated collection of logically related records or files. A database consolidates records previously stored in separate files into a common pool of data records that provides data for many applications. The data is managed by systems software called database management systems (DBMS). The data stored in a database is independent of the application programs using it and of the types of secondary storage devices on which it is stored.

## Buffer Management

A DBMS must manage a huge amount of data, and in the course of processing the required space for the blocks of data will often be greater than the memory space available. For this there is the need to manage a memory in which to load and unload the blocks. The buffer manager is responsible primarily for managing the operations inherent saving and loading of the blocks.

In fact, the operations that provide the buffer manager are these:

* FIX: This command tells the operator of the buffer to load a block from disk and return the pointer to the memory where it is loaded. If the block was already in memory, the buffer manager needs only to return the pointer, otherwise he must load from disk and bring it into memory. If the buffer memory is full but it is possible to have 2 situations: or the possibility of releasing a portion of memory that is occupied by transactions already completed. In this case, before freeing the area the content is written to disk if any block of this area had been changed.
* There is the possibility of free memory to be occupied because transitions still ongoing. In this case, the buffer manager can work in 2 ways:
  + In the first mode (STEAL), the operator of the free buffer memory occupied by a transition already active, possibly saving your changes to disk.
  + In the second mode (NOT STEAL), the transition requested block is made to wait until the free memory.
* SET DIRTY: invoking this command, you mark a block of memory as amended. Before introducing the last 2 commands you need to anticipate that the DMBS can operate in 2 modes: Force and NOT FORCE. When working in FORCE mode, the rescue disk is in synchronous mode with the commit of a transaction. When working mode is NOT FORCE the rescue is carried out from time to time in asynchronous manner. Typically, commercial database operating mode NOT FORCE because this allows an increase in performance: the block may undergo multiple changes in memory before being saved, then you can choose to make the saves when the system is unloading.
* Force: This command will cause the operator of the buffer to make the writing in synchronously with the completion (commit) the transaction
* FLUSH: This command will cause the operator of the buffer to perform the rescue,when in how NOT FORCE.

## Transaction Log

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. While the DBMS executes transactions that modify the database, it also automatically updates the transaction log. The transaction log stores:

* A record for the beginning of the transaction.
* For each transaction component (SQL statement):
* The type of operation being performed (updates, delete, insert).
* The names of the objects affected by the transaction (the name of the table).
* The “before” and “after” values for the fields being updated.
* Pointers to the previous and next transaction log entries for the same transaction.
* The ending (COMMIT) of the transaction. Data updates

## Data updates

Immediate Update: As soon as a data item is modified in cache, the disk copy is updated.

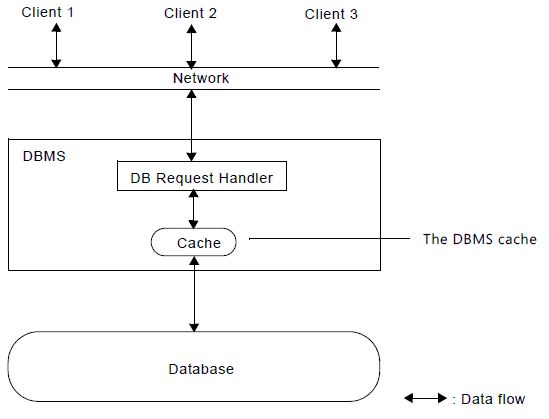
Deferred Update: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.

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

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.



## Transaction Roll back (undo) and Roll Forward

**The Roll back** transaction is a transaction which rolls back the transaction to the beginning of the transaction (Rollback Transaction\_name). It is possible to use before Commit transaction.A rollback is the operation of restoring a database to a previous state by canceling a specific transaction or transaction set. Rollbacks are either performed automatically by database systems or manually by users. When a database user changes a data field but has not yet saved the change, the data is stored in a temporary state or transaction log. Users querying the unsaved data see the unchanged values. The action of saving the data is a commit; this allows subsequent queries for this data to show the new values.

However, a user may decide not to save the data. Under this condition, a rollback command manipulates the data to discard any changes made by the user, and does so without communicating this to the user. Thus, a rollback occurs when a user begins changing data, realizes the wrong record is being updated and then cancels the operation to undo any pending changes. Rollbacks also may be issued automatically after a server or database crash, e.g. after a sudden power loss. When the database restarts, all logged transactions are reviewed; then all pending transactions are rolled back, allowing users to reenter and save appropriate changes.

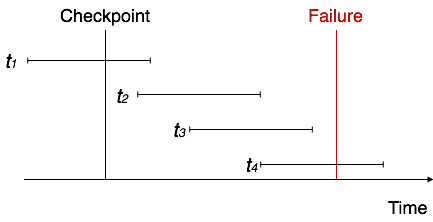
**The Roll forward** is redoing the changes made by a transaction that is after the committed transaction and to over-write the changed value once again to ensure the consistency. Roll Forward occurs when the database restarts after an abnormal shutdown. Its a process of going to the log files and applying changes from the log files to the underlying database. In the case where the underlying tables have been restored from an old backup this can involve millions of updates and take several hours

## Check pointing, Shadow Paging

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 .

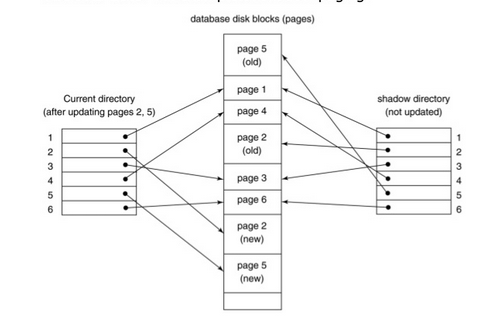


* The recovery system reads the logs backwards from the end to the last checkpoint.
* It maintains two lists, an undo-list and a redo-list.
* If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
* If the recovery system sees 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

In computer science, shadow paging is a technique for providing atomicity and durability (two of the ACID properties) in database systems. A page in this context refers to a unit of physical storage (probably on a hard disk), typically of the order of 210 to 216 bytes.

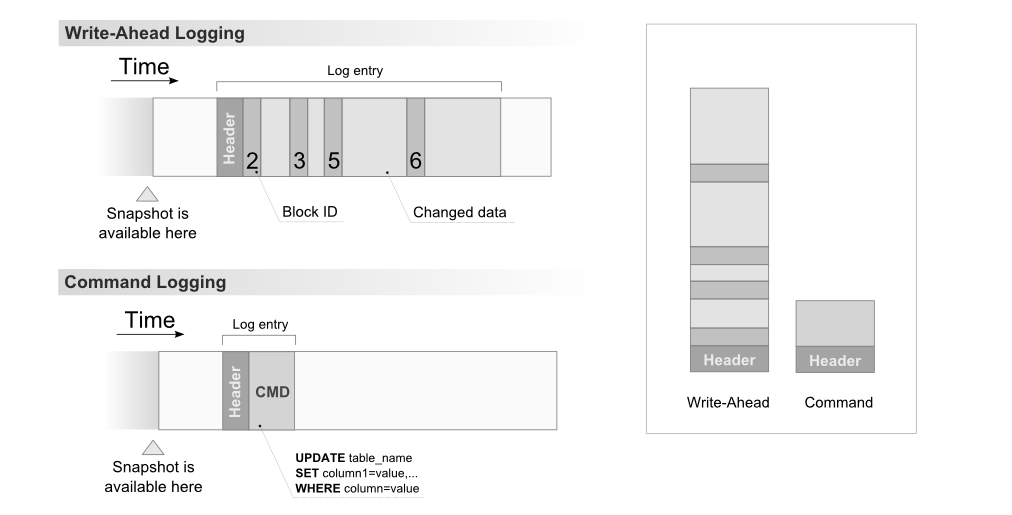
**Shadow paging** is a copy-on-write technique for avoiding in-place updates of pages. Instead, when a page is to be modified, a shadow page is allocated. Since the shadow page has no references (from other pages on disk), it can be modified liberally, without concern for consistency constraints, etc. When the page is ready to become durable, all pages that referred to the original are updated to refer to the new replacement page instead. Because the page is "activated" only when it is ready, it is atomic.



## Recovery Schemas (WAL: Write Ahead Logging Protocol)

Write-Ahead Logging (WAL) is a standard method for ensuring data integrity. Briefly, WAL's central concept is that changes to data files (where tables and indexes reside) 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. (This is roll-forward recovery, also known as REDO.)

WAL protocol Write-ahead logging (WAL) is a family of techniques for providing atomicity and durability (two of the ACID properties) in database systems. In a system using WAL, all modifications are written to a log before they are applied. Usually both redo and undo information is stored in the log. Before a block of data in main memory can be output to the database (in nonvolatile storage), all log records pertaining to data in that block must have been output to stable storage. This rule is called the WAL rule. Strictly speaking, the WAL rule requires only that the undo information in the log have been output to stable storage, and permits the redo information to be written later. The difference is relevant in systems where undo information and redo information are stored in separate log records**.** Write-ahead logging is employed to flush log records to the persistent log file before data pages are written or at commit time**.** Imagine a program that is in the middle of performing some operation when the machine it is running on loses power. Upon restart, that program might well need to know whether the operation it was performing succeeded, half-succeeded, or failed. If a write-ahead log were used, the program could check this log and compare what it was supposed to be doing when it unexpectedly lost power to what was actually done. On the basis of this comparison, the program could decide to undo what it had started, complete what it had started, or keep things as they are.



## Failure with Loss of No 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 multi database transaction involving each has concluded, the coordinator sends a message "prepare for commit" to each participant to get ready for committing the transaction. Each participating database receiving that message will force-write all log records and needed information for local recovery to disk and then send a "ready to commit" or "OK" signal to the coordinator. If the force-writing to disk fails or the local transaction cannot commit for some reason, the participating database sends a "cannot commit" or "not OK" signal to the coordinator. If the coordinator does not receive a reply from a database within a certain time out interval, it assumes a "not OK" response.

• **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. On the other hand, if one or more of the participating databases or the coordinator have a "not OK" response, the transaction has failed, and the coordinator sends a message to "roll back" or UNDO the local effect of the transaction to each participating database. This is done by undoing the transaction operations, using the log.