ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**DBMS Lab Assignment #9**

**Submitted by:**

Suman Malla

013BSCCSIT041

**Submitted to:**

|  |  |
| --- | --- |
| Er. Sanjay K. Yadav |  |

Lecturer  
Department of Computer Science

Date of Submission: Friday, October 02, 2015

**1) Database recovery**

In computing, data recovery is a process of salvaging inaccessible data from corrupted or damaged secondary storage, removable media or files, when the data they store cannot be accessed in a normal way. The data is most often salvaged from storage media such as internal or external hard disk drives (HDDs), solid-state drives (SSDs), USB flash drives, magnetic tapes, CDs, DVDs, RAID subsystems, and other electronic devices. Recovery may be required due to physical damage to the storage device or logical damage to the file system that prevents it from being mounted by the host operating system (OS).The most common data recovery scenario involves an operating system failure, malfunction of a storage device, accidental damage or deletion, etc. (typically, on a single-drive, single-partition, single-OS system), in which case the goal is simply to copy all wanted files to another drive. This can be easily accomplished using a Live CD, many of which provide a means to mount the system drive and backup drives or removable media, and to move the files from the system drive to the backup media with a file manager or optical disc authoring software. Such cases can often be mitigated by disk partitioning and consistently storing valuable data files (or copies of them) on a different partition from the replaceable OS system files.

Another scenario involves a drive-level failure, such as a compromised file system or drive partition, or a hard disk drive failure. In any of these cases, the data cannot be easily read. Depending on the situation, solutions involve repairing the file system, partition table or master boot record, or drive recovery techniques ranging from software-based recovery of corrupted data, hardware- and software-based recovery of damaged service areas (also known as the hard disk drives "firmware"), to hardware replacement on a physically damaged drive. If a drive recovery is necessary, the drive itself has typically failed permanently, and the focus is rather on a one-time recovery, salvaging whatever data can be read.

In a third scenario, files have been "deleted" from a storage medium. Typically, the contents of deleted files are not removed immediately from the drive; instead, references to them in the directory structure are removed, and the space they occupy is made available for later overwriting. For the end users, deleted files are not discoverable through a standard file manager, but that data still technically exists on the drive. In the meantime, the original file contents remain, often in a number of disconnected fragments, and may be recoverable.The term "data recovery" is also used in the context of forensic applications or espionage, where data which have been encrypted or hidden, rather than damaged, are recovered.

**2) Purpose Of Data Recovery**

As a backup administrator, your principal duty is to devise, implement, and manage a backup and recovery strategy. In general, the purpose of a  strategy is to protect the database against data loss and reconstruct the database after data loss. Typically, backup administration tasks include the following:

* Planning and testing responses to different kinds of failures
* Configuring the database environment for backup and recovery
* Setting up a backup schedule
* Monitoring the backup and recovery environment
* Troubleshooting backup problems
* Recovering from data loss if the need arises
* Application Errors
* Data Protection
* Media Failures

**3) Types of Failure**

Transaction failure: Individual transactions fail

Logical error: Internal problem within the transaction

System error: External problem during transaction execution (e.g., deadlock)

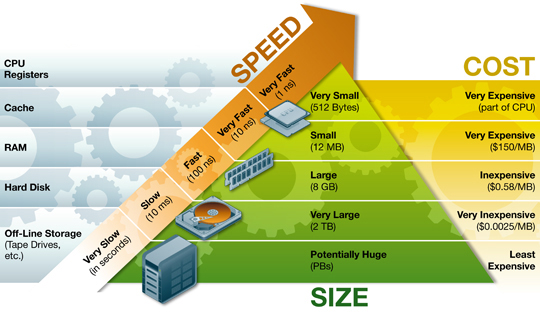
System crash: Problem with overall database server execution; terminates the current process

Fail-stop assumption: Data in non-volatile storage is unharmed in the event of a system crash

Disk failure: Problem with storage media

**4) The storage hierarchy**

The range of memory and storage within and attached to a computer system is known as the Storage Hierarchy and to help understand this further can be categorised into 4 segments. As memory and storage devices move down the hierarchy they reduce in performance and cost/MB or GB but tend to rise in capacity through to the last category which includes removable media which in effect has no restriction on capacity a device can store.[4]



* **Primary Storage** is the top level and is made up of CPU registers, CPU cache and memory which are the only components that are directly accessible to the systems CPU. The CPU can continuously read data stored in these areas and execute all instructions as required quickly in a uniform manner. Secondary Storage differs from primary storage in that it is not directly accessible by the CPU. A system uses input/output (I/O) channels to connect to the secondary storage which control the data flow through a system when required and on request
* **Secondary storage** is non-volatile so does not lose data when it is powered down so consequently modern computer systems tend to have a more secondary storage than primary storage. All secondary storage today consist of hard disk drives (HDD), usually set up in a RAID configuration, however older installations also included removable media such us magneto optical or MO
* **Tertiary Storage** is mainly used as backup and archival of data and although based on the slowest devices can be classed as the most important in terms of data protection against a variety of disasters that can affect an IT infrastructure. Most devices in this segment are automated via robotics and software to reduce management costs and risk of human error and consist primarily of disk & tape based back up devices
* **Offline Storage** is the final category and is where removable types of storage media sit such as tape cartridges and optical disc such as CD and DVD. Offline storage is can be used to transfer data between systems but also allow for data to be secured offsite to ensure companies always have a copy of valuable data in the event of a disaster.

**5) Buffer Management**

Database buffer can be implemented either in an area of real main-memory reserved for the database, or in virtual memory Implementing buffer in reserved main-memory has drawbacks. Memory is partitioned before-hand between database buffer and applications, limiting flexibility. Needs may change, and although operating system knows best how memory should be divided up at any time, it cannot change the partitioning of memory. Database buffers are generally implemented in virtual memory inspite of some drawbacks: When operating system needs to evict a page that has been modified, to make space for another page, the page is written to swap space on disk. When database decides to write buffer page to disk, buffer page may be in swap space, and may have to be read from swap space on disk and output to the database on disk, resulting in extra I/O Known as dual paging problem. Ideally when swapping out a database buffer page, operating

System should pass control to database, which in turn outputs page to database instead of to swap space (making sure to output log records first) Dual paging can thus be avoided, but common operating systems do not support such functionality.

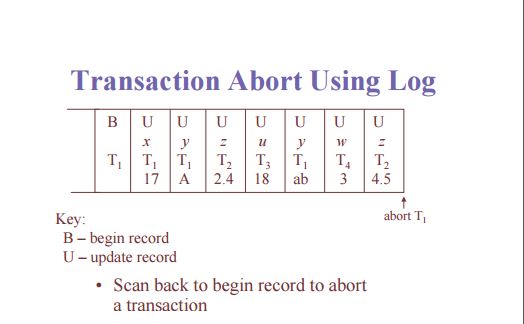
**6) Transaction Log**

Scan log backwards using tid to identify transaction’s update records

• Reverse each update using before image

• In a strict system, new values are unavailable to concurrent transactions (as a result of long term exclusive locks); hence rollback makes transaction atomic

• Problem: terminating scan (log can be long) • Solution: append begin record containing tid prior to first update record.



**7) 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.

**8) 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.

**9) Transaction Roll back (Undo) and Roll Forward**

In computer science, transaction processing is information processing that is divided into individual, indivisible operations called transactions. Each transaction must succeed or fail as a complete unit; it can never be only partially complete.

The basic principles of all transaction-processing systems are the same. However, the terminology may vary from one transaction-processing system to another, and the terms used below are not necessarily universal.

* **Rollback(undo)**

Transaction-processing systems ensure database integrity by recording intermediate states of the database as it is modified, then using these records to restore the database to a known state if a transaction cannot be committed. For example, copies of information on the database prior to its modification by a transaction are set aside by the system before the transaction can make any modifications (this is sometimes called a before image). If any part of the transaction fails before it is committed, these copies are used to restore the database to the state it was in before the transaction began.

* **Roll forward**

It is also possible to keep a separate journal of all modifications to a database management system. (Sometimes called after images). This is not required for rollback of failed transactions but it is useful for updating the database management system in the event of a database failure, so some transaction-processing systems provide it. If the database management system fails entirely, it must be restored from the most recent back-up. The back-up will not reflect transactions committed since the back-up was made. However, once the database management system is restored, the journal of after images can be applied to the database (roll forward) to bring the database management system up to date. Any transactions in progress at the time of the failure can then be rolled back. The result is a database in a consistent, known state that includes the results of all transactions committed up to the moment of failure.

**10) Shadow paging**

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.

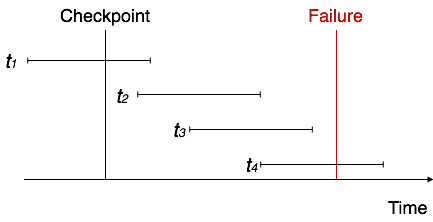
Shadow paging is similar to the old master–new master batch processing technique used in mainframe database systems. In these systems, the output of each batch run (possibly a day's work) was written to two separate disks or other form of storage medium. One was kept for backup, and the other was used as the starting point for the next day's work.

* **Check pointing**

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.

**11) Recovery Schemes (WAL: write ahead logging protocol)**

In computer science, 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.

The purpose of this can be illustrated by an example. 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 is used, the program can 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.

WAL allows updates of a database to be done in-place. Another way to implement atomic updates is with shadow paging, which is not in-place. The main advantage of doing updates in-place is that it reduces the need to modify indexes and block lists.

**12) Failure with loss of non-volatile storage (General Concept)**

So far we assumed no loss of non-volatile storage. Technique similar to check pointing used to deal with loss of non-volatile 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 check pointing 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 to log on stable storage. ! To recover from disk failure restore database from most recent dump. Consult the log and redo all transactions that committed after the dump. Can be extended to allow transactions to be active during dump known as fuzzy dump or online dump.

**13) Recovery in Multi-database system**

A multi database transaction requires access to multiple databases. The DBs may even be stored on different types of DBMS.

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.

The net effect of the two-phase commit protocol is that either all participating databases commit the effect of the transaction or none of them do. In case any of the participants—or the coordinator—fails, it is always possible to recover to a state where either the transaction is committed or it is rolled back. A failure during or before Phase 1 usually requires the transaction to be rolled back, whereas a failure during Phase 2 means that a successful transaction can recover and commit.

**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