ST. XAVIER'S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**DATABASE MANAGEMENT SYSTEM**

**THEORY ASSIGNMENT # 9**

**SUBMITTED BY:**

Dikita Tualdhar  
013BSCCSIT018

2nd Year/4th Sem

**SUBMITTED TO:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav**  Supervisor,  Department of Computer Science  St. Xavier’s College |  |

**Date of Submission:** 4th October, 2015

1. **Data recovery**
   1. **Purpose of Data Recovery**

Data recovery is the process of restoring data that has been lost, accidentally deleted, corrupted or made inaccessible for any reason. Data recovery typically refers to the restoration of data to a desktop, laptop, server, or external storage system from a backup.

The data recovery process may vary, depending on the circumstances of the data loss, the data recovery software used to create the backup, and the backup target media. For example, many desktop and laptop backup software platforms allow end users to restore lost files themselves, while restoration of a corrupted database from a tape backup is a more complicated process that requires IT intervention. Data recovery can also be provided as service. Such services are typically used to retrieve important files that were not backed up and accidentally deleted from a computer's file system but still remain on disk in fragments.

An organization's disaster recovery plan should make known that in the organization is responsible for recovering data, provide a strategy for how data will be recovered and document acceptable recovery point and recovery time objectives.

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.

* 1. **Types of Failure**

1. Transaction failures:

It causes a transaction to fail. It is again of 2 types.

1. Logical error: When transaction can no longer continue with its normal execution because of some internal condition such as bad i/p, data not found, data overflow etc. it gives incorrect result. This is logical error. Eg: any student data.
2. System error: When system enters in some undesirable state, then also the transaction can continue its normal execution. Eg: deadlock state.
3. System crash:

In system crash there is a h/w malfunction or  
a bug in the database s/w or the operating system, that causes  
the loss of content of volatile storage and brings transaction  
processing to a halt.

1. Disk failure:

In this, a disk block loses its content as a result of either head crash or failure during data transfer operation. Copies of data on other disk as tapes are used to recover them from failure.

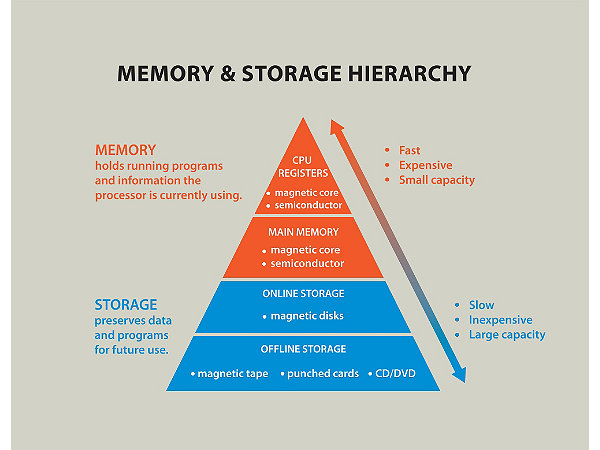
|  |  |
| --- | --- |
| TYPES | CAUSES |

|  |  |
| --- | --- |
| 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. |

* 1. **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.

1. **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
2. **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
3. **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
4. **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.



* 1. **Buffer Management**

The primary purpose of a SQL Server database is to store and retrieve data, so intensive disk I/O is a core characteristic of the Database Engine. And because disk I/O operations can consume many resources and take a relatively long time to finish, SQL Server focuses on making I/O highly efficient. Buffer management is a key component in achieving this efficiency. The buffer management component consists of two mechanisms: the buffer manager to access and update database pages, and the buffer cache (also called the buffer pool), to reduce database file I/O.

A buffer is an 8-KB page in memory, the same size as a data or index page. Thus, the buffer cache is divided into 8-KB pages. The buffer manager manages the functions for reading data or index pages from the database disk files into the buffer cache and writing modified pages back to disk. A page remains in the buffer cache until the buffer manager needs the buffer area to read in more data. Data is written back to disk only if it is modified. Data in the buffer cache can be modified multiple times before being written back to disk.

When SQL Server starts, it computes the size of virtual address space for the buffer cache based on a number of parameters such as the amount of physical memory on the system, the configured number of maximum server threads, and various startup parameters. SQL Server reserves this computed amount of its process virtual address space (called the memory target) for the buffer cache, but it acquires (commits) only the required amount of physical memory for the current load. You can query the bpool\_commit\_target and bpool\_committed columns in the sys.dm\_os\_sys\_info catalog view to return the number of pages reserved as the memory target and the number of pages currently committed in the buffer cache, respectively.

The interval between SQL Server startup and when the buffer cache obtains its memory target is called ramp-up. During this time, read requests fill the buffers as needed. For example, a single-page read request fills a single buffer page. This means the ramp-up depends on the number and type of client requests. Ramp-up is expedited by transforming single-page read requests into aligned eight-page requests. This allows the ramp-up to finish much faster, especially on machines with a lot of memory.

Because the buffer manager uses most of the memory in the SQL Server process, it cooperates with the memory manager to allow other components to use its buffers. The buffer manager interacts primarily with the following components:

1. Resource manager to control overall memory usage and, in 32-bit platforms, to control address space usage.
2. Database manager and the SQL Server Operating System (SQLOS) for low-level file I/O operations.
3. Log manager for write-ahead logging.
   1. **Transaction Log**

In the field of [databases](https://en.wikipedia.org/wiki/Database) in [computer science](https://en.wikipedia.org/wiki/Computer_science), a transaction log (also transaction journal, database log, binary log or audit trail) is a history of actions executed by a [database management system](https://en.wikipedia.org/wiki/Database_management_system) to guarantee [ACID](https://en.wikipedia.org/wiki/ACID) properties over [crashes](https://en.wikipedia.org/wiki/Crash_(computing)) or hardware failures. Physically, a log is a [file](https://en.wikipedia.org/wiki/Computer_file) listing changes to the database, stored in a stable storage format.

If, after a start, the database is found in an [inconsistent](https://en.wikipedia.org/wiki/Consistency_(database_systems)) state or not been shut down properly, the database management system reviews the database logs for [uncommitted](https://en.wikipedia.org/wiki/Commit_(data_management)) transactions and [rolls back](https://en.wikipedia.org/wiki/Rollback_(data_management)) the changes made by these [transactions](https://en.wikipedia.org/wiki/Database_transaction). Additionally, all transactions that are already committed but whose changes were not yet materialized in the database are re-applied. Both are done to ensure [atomicity](https://en.wikipedia.org/wiki/Atomicity_(database_systems)) and [durability](https://en.wikipedia.org/wiki/Durability_(computer_science)) of transactions.

This term is not to be confused with other, human-readable [logs](https://en.wikipedia.org/wiki/Data_logging#Computer_data_logging) that a database management system usually provides.

In [computer storage](https://en.wikipedia.org/wiki/Computer_storage), a journal is a [chronological](https://en.wikipedia.org/wiki/Chronology) record of [data processing](https://en.wikipedia.org/wiki/Data_processing) operations that may be used to construct or reinstate an historical or alternative version of a [computer system](https://en.wikipedia.org/wiki/Computer_system) or [computer file](https://en.wikipedia.org/wiki/Computer_file). In database management systems, a journal is the record of data altered by a given process.

* 1. **Data Updates**

The update statement is used to change or modify the existing records in a database table. It is typically used in conjugation with the where clause to apply the changes to only those records that matches specific criteria.

The basic syntax of the update statement can be given with:

UPDATE table\_name SET column1=value, column2=value2,... where column\_name=some\_value

Let's make a SQL query using the update statement and WHERE clause, after that we will execute this SQL query through passing it to the mysqli\_query() function to update the tables records. Consider the following "persons" table inside the "demo" database:

+-----------+------------+-----------+----------------------+

| person\_id | first\_name | last\_name | email\_address |

+-----------+------------+-----------+----------------------+

| 1 | Peter | Parker | peterparker@mail.com |

| 2 | John | Rambo | johnrambo@mail.com |

| 3 | Clark | Kent | clarkkent@mail.com |

| 4 | John | Carter | johncarter@mail.com |

| 5 | Harry | Potter | harrypotter@mail.com |

+-----------+------------+-----------+----------------------+

* 1. **Data Caching**

Caching in applications is going to vary from one platform to another. An object cache is a mechanism that you can use to put commonly used objects into memory so that you don't need to pay the cost to retrieve the data and recreate them.

Many applications today are being developed and deployed on multi-tier environments that involve browser-based clients, web application servers and backend databases. These applications need to generate web pages on-demand by talking to backend databases because of their dynamic nature, making middle-tier database caching an effective approach to achieve high scalability and performance.

In three tier architecture, the application tier and data tier can be in different hosts. Throughput of the application is affected by the network speed. This network overhead will be avoided by having the database at the application tier. As commercial databases are heavy weight, it is not practically feasible to have the application and the database at the same host. There are lot of light-weight databases available on the market, which can be used to cache data from the commercial databases. The benefits of data caching are as follows:

1. Scalability: distribute query workload from backend to multiple cheap front-end systems.
2. Flexibility: achieve QoS, where each cache hosts different parts of the backend data, e.g., the data of Platinum customers are cached while that of ordinary customers is not.
3. Availability: by continued service for applications that depend only on cached tables even if the backend server is unavailable.
4. Performance: by potentially responding fast because of locality of data and smoothing out load peaks by avoiding round-trips between middle-tier and data-tier
   1. **Transition Roll Back (Undo) and Roll Forward**

In [database](https://en.wikipedia.org/wiki/Database) technologies, a rollback is an operation which returns the database to some previous state. Rollbacks are important for database [integrity](https://en.wikipedia.org/wiki/Data_integrity), because they mean that the database can be restored to a clean copy even after erroneous operations are performed. They are crucial for recovering from database server crashes; by rolling back any [transaction](https://en.wikipedia.org/wiki/Database_transaction) which was active at the time of the crash, the database is restored to a consistent state. Rollback is the process of undoing changes and reverting to a previous state. This usually occurs either on request when a program detects some logical error and decides the transaction should not take place, or, when the DBMS loses contact with the program before an explicit "COMMIT" has been requested.

The rollback feature is usually implemented with a [transaction log](https://en.wikipedia.org/wiki/Database_log), but can also be implemented via [multi version concurrency control](https://en.wikipedia.org/wiki/Multiversion_concurrency_control).

In SQL, rollback is a command that causes all data changes since the last begin work, or start transaction to be discarded by the relational database management systems (RDBMS), so that the state of the data is "rolled back" to the way it was before those changes were made.

A rollback statement will also release any existing save points that may be in use.

In most SQL dialects, rollbacks are connection specific. This means that if two connections are made to the same database, a rollback made in one connection will not affect any other connections. This is vital for proper concurrency.

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. It’s 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.

* 1. **Check Pointing, Shadow Paging**

**Check pointing** is a technique to add fault tolerance into computing systems. It basically consists of saving a snapshot of the application's state, so that it can restart from that point in case of failure. This is particularly important for long running application that are executed in vulnerable computing system. It is a kind of entry in database log. It is used to restore the database in the consistent state. In case of failure, the recovery manager requires that the entire log be examined to process recovery→ time consuming. A quick way to limit the amount of log to scan on recovery can be achieved using checkpoints. A [checkpoint] 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. Hence, all transactions with [commit, T] entry in the log before [checkpoint] entry do not need to have their WRITE operations redone in case of crash. Since all their update have been recorded in the database on the disk during check pointing. The recovery manager must decides at what intervals to take a checkpoint. The intervals measured in time, say every m minutes. Or the intervals measured in the number of committed transactions since the last checkpoint, say t transactions where m & t are system parameters.

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.

If the referring pages must also be updated via shadow paging, this procedure may recourse many times, becoming quite costly. One solution, employed by the WAFL file system (Write Anywhere File Layout) is to be lazy about making pages durable (i.e. write-behind caching). This increases performance significantly by avoiding many writes on hotspots high up in the referential hierarchy (e.g.: a file system superblock) at the cost of high commit latency.

Write-ahead logging is a more popular solution that uses in-place updates.[citation needed]

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.

Shadow paging is also similar to purely functional data structures, in that in-place updates are avoided.

* 1. **Recovery Schemes (WAL: Write Ahead Logging Protocol)**

Two types of log entry –log record- information for a write command. The information needed for UNDO. A UNDO-type log entries including the old values (BFIM). Since this is needed to undo the effect of the operations from the log. The information needed for REDO. REDO-type log entries including the new values (AFIM). Since it is needed to redo the effect of the operations from the log In UNDO/REDO algorithms, both types of log entries are combined. The log includes read commands only when cascading rollback is possible Write-Ahead Logging (WAL) is the fundamental rule??? That ensures that a record –entry- of every change to the DB is available while attempting to recover from a crash. Suppose that that the BFIM of a data item on disk has been overwritten by the AFIM on disk and a crash occurs. Without ensuring that this BFIM is recorded in the appropriate log entry and the log is flushed to disk before the BFIM is overwritten with the AFIM in the DB on disk, the recovery will not be possible. Suppose a transaction made a change and committed with some of its changes not yet written to disk? Without a record of these changes written to disk, there would be no way to ensure that the changes of the committed transaction survive crashes The log is a simple sequential (append-only) file When a log record is written, it is stored in the current log in the DBMScache and after written to disk as soon as is feasible. With Write-Ahead Logging, the log blocks containing 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. WAL protocol for a recovery algorithm that requires both: UNDO and REDO.

1. The before image of an item cannot be overwritten by its after image in the database on disk until all UNDO-type log records for the updating transaction up to this point in time- have been force-written to disk.
   * Ensures atomicity.
2. The commit operation of a transaction cannot be completed until all the REDO-type and UNDO-type log records for that transaction have been force-written to disk.
   * Ensures durability.
   * Commercial DBMSs and WAL

IBM DB2, Informix, Microsoft SQL Server, Oracle 8, and Sybase ASE all use a WAL scheme for recovery.

* To facilitate the recovery process, the DBMS recovery subsystem may need to maintain a number of lists.
* List of active transactions: transactions started but not committed yet
* List of committed transactions since last checkpoint.
* List of aborted transactions since last checkpoint.
  1. **Failure with Loss of Non-Volatile Storage (General Concepts)**

Technique similar to check pointing used to deal with loss of non-volatile storage

1. Periodically dump the entire content of the database to stable storage
2. No transaction may be active during the dump procedure; a procedure similar to check pointing must take place
3. Output all log records currently residing in main memory onto stable storage.
4. Output all buffer blocks onto the disk.
5. Copy the contents of the database to stable storage.
6. Output a record <dump> to log on stable storage.
   1. **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. Some DBMS may be relational, whereas others are object-oriented, etc. Each DBMS involved in the multi database transaction may have its own recovery technique and transaction manager separate from those of the other DBMSs. Use a two-level recovery mechanism to maintain the atomicity of a multi database transaction.

* A global recovery manager or coordinator.
* The local recovery managers.

The coordinator usually follows a two-phase commit protocol.

* Phase 1

When all participating databases signal the coordinator that the part of the multi database transaction has concluded, the coordinator sends a message «prepare to 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 or «cannot commit» -or not OK- if it fails for some other reasons.

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 the participants DB reply «OK» and also the coordinator, the transaction is successful and the coordinator sends a «commit» signal for the transaction to the participant databases.

Each participant database completes transaction commit by writing a [commit] entry for the transaction in the log and permanently updating the database if needed.

If one or more participating DBs or the coordinator sends «not OK» message, the transaction fails and the coordinator sends a message to «rollback» -or UNDO- the local effect of the transaction to each participating database.

The UNDO of the local effect is done by using the log at each participating database.