**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.

**Purpose of data recovery**

* 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

**Types of failure**

* Transaction failures
  + overflow, interrupt, data not available, explicit rollback, concurrency enforcement, programming errors
  + no memory loss.
* Process Failure
  + failure in a user, server, or background process of a database instance such as an abnormal disconnect or process termination
* Network Failure
  + disconnected from client workstations to database servers, or to several database servers to form a distributed database system
  + network failures such as aborted phone connections or network communication software failures can interrupt the normal operation of a database system
* System crashes
  + due to hardware or software errors
  + main memory content is lost
* Media failures
  + problems with disk head, unreadable media surface(parts of ) information on secondary storage may be lost
* Natural disasters
  + fire, flood, earthquakes, theft, etc.
  + physical loss of all information on all media

**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:

* Bits (characters)
* Fields
* Records
* Files
* Databases

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**

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.

The subsystem responsible for the allocation of buffer space is called the buffer manager

The buffer manager must use some sophisticated techniques in order to provide good service:

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.

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.

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.

LRU is acceptable in operating systems, however, a database system is able to predict future references more accurately

**Transaction Log**

Every SQL Server 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.

**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 rollback (Undo) and Roll forward**

The process of applying logged changes to data in a database to bring the data forward in time is known as rolling forward. The set of all data restored is called the roll forward set. A roll forward set is defined by restoring one or more full backups, such as a database or partial backup or a set of file backups. If a RESTORE statement specifies filegroups, files, or pages, only these items are included in the roll forward set. Otherwise, all files in the backup being restored are included in the roll forward set. If the full backup contains log records, the restored data will be rolled forward using this log.

After the redo phase has rolled forward all the log transactions, a database typically contains changes made by transactions that are uncommitted at the recovery point. This makes the rolled forward data transactionally inconsistent. The recovery process opens the transaction log to identify uncommitted transactions. Uncommitted transactions are undone by being rolled back, unless they hold locks that prevent other transactions from viewing transactionally inconsistent data. This step, is called the undo (or roll back) phase. If the data is already transactionally consistent at the start of the recovery process, the undo phase is skipped. After the database is transactionally consistent, recovery brings the database online.

After one or more backups have been restored, recovery typically includes both the redo and undo phases. Every full and differential backup contains enough transaction log records to allow for the data in that backup to be recovered to a self-consistent state.

**Check pointing, shadow paging**

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:

* Suspend execution of transactions temporarily.
* Force write modified buffer data to disk.
* Write a [checkpoint] record to the log, save the log to disk.
* Resume normal transaction execution.

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

**Shadow Paging**

* Maintain two page tables during life of a transaction: current page and shadow page table.
* When transaction starts, two pages are the same.
* Shadow page table is never changed thereafter and is used to restore database in event of failure.
* During transaction, current page table records all updates to database.
* When transaction completes, current page table becomes shadow page table.

**Recovery schemes (WAL: Write Ahead Logging Protocol)**

Recovery scheme responsible for handling failures and restoring database to consistent state

Recovery algorithm has two parts

* Actions taken during normal operation to ensure system can recover from failure (e.g., backup, log file)
* Actions taken after a failure to restore database to consistent state

Write-Ahead Logging (WAL)

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.

* A 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.

WAL protocol for a recovery algorithm that requires both UNDO and REDO.

* 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.
* 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.

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

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 periodicallydumped 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 multidatabase system**

A multidatabase transaction require access to multiple databases.The DBs may even be stored on different types of DBMS.

To maintain the atomicity of a multidatabase 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. 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.