# Database Recovery

There are many situations in which a transaction may not reach a commit or abort point.

* 1. An operating system crash can terminate the DBMS processes
  2. The DBMS can crash
  3. The system might lose power
  4. A disk may fail or other hardware may fail.
  5. Human error can result in deletion of critical data.

In any of these situations, data in the database may become inconsistent or lost.

For example, if a transaction has completed 30 out of 40 scheduled writes to the database when the DBMS crashes, then the database may be in an inconsistent state as only part of the transaction’s work was completed.

**Database Recovery** is the process of restoring the database and the data to a consistent state. This may include restoring lost data up to the point of the event (e.g. system crash).

# Purpose of Data Recovery

* In most cases this just isn't the fact! Whatever happened to your data - whether files were accidentally deleted, a virus has wiped out the boot record, the drive was formatted or f disked or even if it is no longer recognized by the operating system, as long as it wasn't physically overwritten, the data which was on the drive is still there.
* The files just aren't accessible anymore the way they should be. For example, if you delete a file, the file's data is not deleted from the drive, but instead a signature byte is set at the start of the file's file name. This signature byte tells the operating systems that this area can be overwritten by other data next. And that's exactly what happens. The next time you write something to the drive, the new data will be written to the so marked area. But this also means, that as long as nothing new is written to the drive, the data is still there, intact and can therefore be recovered.
* This behavior is also the reason why trying to undelete single deleted files often isn't successful. When you delete a file -and you empty the recycle bin as well- as soon as you notice that you still need the file, chances are that you have done something in the meantime, that has overwritten the data area of the file.
* Even in cases which seem the most radical - when you i.e. have formatted your drive from FAT32 to NTFS- and five minutes later realize that you didn't mean to format THIS drive, all of your files are still there. When you format a FAT32 drive, everything is destroyed (the boot record, the FAT, the root directory) except for the partition table and the data. And that's all you need! As long as the data's still there your files can be reconstructed - very often nearly perfectly.

# Types of Failure

The major types of failures involving data integrity (as opposed to data security) are:

•

Transaction Failure

Logical error

. The transaction cannot continue with its normal executionbecause of such things as bad input, data not found, or resource limitexceeded.

System error

The system has entered an undesirable state (for example,deadlock), as a result of which a transaction cannot continue with itsnormal execution. The transaction, however, can be reexecuted at a later time.

System Crash

There is a hardware malfunction, or a bug in the databasesoftware or the operating system, that causes the loss of the content of volatilestorage, and brings transaction processing to a halt. The content of the nonvolatilestorage remains intact, and is not corrupted.

Disk Failure.

A disk block loses its contents as a result of either a head crash or failure during a data transfer. Copies of data on other disks, or archival backupson tertiary media, such as tapes, are used to recover from the failure

# 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.

# Buffer Management

* + We need to use disk storage for the database, and to transfer blocks of data between MM and disk.
  + We also want to minimize the number of such transfers, as they are time-consuming.
  + One way is to keep as many blocks as possible in MM.
  + Usually, we cannot keep all blocks in MM, so we need to manage the allocation of available MM space.
  + The **buffer** is the part of MM available for storage of **copies** of disk blocks.
  + The subsystem responsible for the allocation of buffer space is called the **buffer manager**.
  + The buffer manager handles all requests for blocks of the database.
  + If the block is already in MM, the address in MM is given to the requestor.
  + If not, the buffer manager must read the block in from disk (possibly displacing some other block if the buffer is full) and then pass the address in MM to the requestor.

1. The buffer manager must use some sophisticated techniques in order to provide good service:
   * **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.
   * **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.
2. **Replacement Strategy:** 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.
   * **General Assumption:** Blocks referenced recently are likely to be used again.
   * **Therefore:** if we need space, throw out the least recently referenced block. (LRU replacement scheme)
3. LRU is acceptable in **operating systems**, however, a database system is able to predict future references more accurately.

# Transaction Log

1. The transaction log is a serial record of all the transactions that have been performed against the database since the transaction log was last backed up. With transaction log backups, you can recover the database to a specific point in time (for example, prior to entering unwanted data), or to the point of failure.
2. When restoring a transaction log backup, Microsoft® SQL Server™ rolls forward all changes recorded in the transaction log. When SQL Server reaches the end of the transaction log, it has re-created the exact state of the database at the time the backup operation started. If the database is recovered, SQL Server then rolls back all transactions that were incomplete when the backup operation started.
3. Transaction log backups generally use fewer resources than database backups. As a result, you can create them more frequently than database backups. Frequent backups decrease your risk of losing data.
4. Restoring a database using both database and transaction log backups works only if you have an unbroken sequence of transaction log backups after the last database or differential database backup. If a log backup is missing or damaged, you must create a database or differential database backup and start backing up the transaction logs again. Retain the previous transaction logs backups if you want to restore the database to a point in time within those backups.
5. The only time database or differential database backups must be synchronized with transaction log backups is when starting a sequence of transaction log backups. Every sequence of transaction log backups must be started by a database or differential database backup.

# 7. Data Caching

Caching is just the practice of storing data in and retrieving data from a high-performance store (usually memory) either explicitly or implicitly.

Let me explain. Memory is faster to access than a file, a remote URL (usually), a database or any other external store of information you like. So if the act of using one of those external resources is**significant** then you may benefit from caching to increase performance.

Knuth once said that premature optimization is the root of all evil. Well, premature caching is the root of all headaches as far as I'm concerned. Don't solve a problem until you **have** a problem. Every decision you make comes at a cost that you'll pay to implement it now and pay again to change it later so the longer you can put off making a deicsion and changing your system the better.

So first **identify that you actually have a problem and where it is**. Profiling, logging and other forms of performance testing will help you here. I can't stress enough how important this step is. The number of times I've seen people "optimize" something that isn't a problem is staggering.

Ok, so you have a performance problem. Say your pages are running a query that takes a long time. If it's a read then you have a number of options:

* Run the query as a separate process and put the result into a cache. All pages simply access the cache. You can update the cached version as often as is appropriate (once a day, once a week, one every 5 seconds, whatever is appropriate);
* Cache transparently through your persistence provider, ORM or whatever. Of course this depends on what technology you're using. Hibernate and Ibatis for example support query result caching;
* Have your pages run the query if the result isn't in the cache (or it's "stale", meaning it is calculated longer ago than the specified "age") and put it into the cache. This has concurrency problems if two (or more) separate processes all decide they need to update the result so you end up running the same (expensive) query eight times at once. You can handle this locking the cache but that creates another performance problem. You can also fall back to concurrency methods in your language (eg Java 5 concurrency APIs).

If it's an update (or updates take place that need to be reflected in your read cache) then it's a little more complicated because it's no good having an old value in the cache and a newer value in the database such that you then provide your pages with an inconsistent view of the data. But broadly speaking there are four approaches to this:

* Update the cache and then queue a request to update the relevant store;
* Write through caching: the cache provider may provide a mechanism to persist the update and block the caller until that change is made; and
* Write-behind caching: same as write-through caching but it doesn't block the caller. The update happens asynchronously and separately; and
* Persistence as a Service models: this assumes your caching mechanism supports some kind of observability (ie cache event listeners). Basically an entirely separate process--unknown to the caller--listens for cache updates and persists them as necessary.

Which of the above methodologies you choose will depend a lot on your requirements, what technologies you're using and a whole host of other factors (eg is clustering and failover support required?).

# 8. Transaction Rollback and Roll Forward

A rollback is the undoing of partly completed [database](http://searchsqlserver.techtarget.com/definition/database) changes when a database [transaction](http://searchcio.techtarget.com/definition/transaction) is determined to have failed. The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for ROLLBACK command is as follows:

ROLLBACK;

## Example:

Consider the CUSTOMERS table having the following records:

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

| ID | NAME | AGE | ADDRESS | SALARY |

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

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

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

Following is the example, which would delete records from the table having age = 25 and then ROLLBACK the changes in the database.

SQL> DELETE FROM CUSTOMERS

WHERE AGE = 25;

SQL> ROLLBACK;

As a result, delete operation would not impact the table and SELECT statement would produce the following result:

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

| ID | NAME | AGE | ADDRESS | SALARY |

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

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

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

 Recovering a database by applying different transactions that recorded in the database log files. It is nothing but re-doing the changes made by a transaction i.e. after the committed transaction and to over write the changed value again to ensure consistency.

# 9. Check Pointing

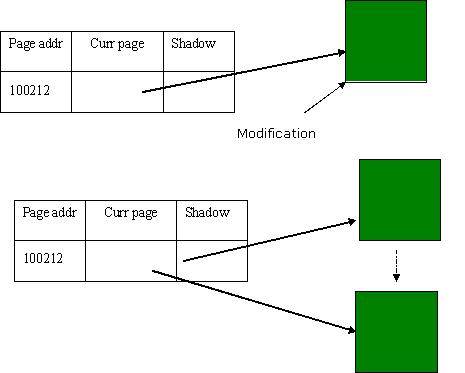
When a checkpoint operation occurs, no matter how it’s triggered (for instance through a manual CHECKPOINT, from a database or differential backup, or automatically) the same set of operations occurs:

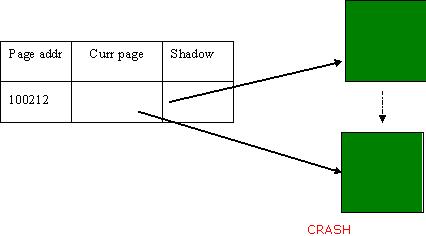
* All dirty data file pages for the database are written to disk (all pages that have changed in memory since they were read from disk or since the last checkpoint), regardless of the state of the transaction that made the change.
* Before a page is written to disk, all log records up to and including the most recent log record describing a change to that page are written to disk (yes, log records can be cached in memory too). This guarantees recovery can work and is called *write-ahead logging*. Log records are written to the log sequentially, and log records from multiple transactions will be interspersed in the log. The log cannot be selectively written to disk, so writing a dirty page to disk that only has a single log record affecting it may mean writing many more previous log records to disk as well.
* Log records describing the checkpoint are generated.
* The LSN of the checkpoint is recorded in the database boot page in the dbi\_checkptLSN field (see [Search Engine Q&A #20: Boot pages, and boot page corruption](http://www.sqlskills.com/blogs/paul/search-engine-qa-20-boot-pages-and-boot-page-corruption/)).
* If in the SIMPLE recovery model, the VLFs in the log are checked to see whether they can be marked inactive (called *clearing* or *truncating* the log – both of which are terrible misnomers, as nothing is either physically cleared or truncated).

Checkpoints are not really tracked in the transaction log – it just serves as a useful repository for information about which transactions are active at the time of the checkpoint. The LSN of the last checkpoint is recorded in the database boot page. This is where recovery starts, and if this page is inaccessible, the database cannot be attached, opened, or processed in any way – partly because it’s the boot page that knows whether the database was cleanly shut down or not, and partly because it’s the only place that records the LSN of the last checkpoint record.

## Shadow Paging

Shadow paging is an alternative to log-based recovery techniques, which has both advantages and disadvantages. It may require fewer disk accesses, but it is hard to extend paging to allow multiple concurrent transactions. The paging is very similar to paging schemes used by the operating system for memory management.  
The idea is to maintain two page tables during the life of a transaction: the current page table and the shadow page table. When the transaction starts, both tables are identical. The shadow page is never changed during the life of the transaction. The current page is updated with each**write**operation. Each table entry points to a page on the disk. When the transaction is committed, the shadow page entry becomes a copy of the current page table entry and the disk block with the old data is released. If the shadow is stored in nonvolatile memory and a system crash occurs, then the shadow page table is copied to the current page table. This guarantees that the shadow page table will point to the database pages corresponding to the state of the database prior to any transaction that was active at the time of the crash, making aborts automatic.





There are drawbacks to the shadow-page technique:

1. **Commit overhead.** The commit of a single transaction using shadow paging requires multiple blocks to be output -- the current page table, the actual data and the disk address of the current page table. Log-based schemes need to output only the log records.
2. **Data fragmentation.** Shadow paging causes database pages to change locations (therefore, no longer contiguous.
3. **Garbage collection.** Each time that a transaction commits, the database pages containing the old version of data changed by the transactions must become inaccessible. Such pages are considered to be garbage since they are not part of the free space and do not contain any usable information. Periodically it is necessary to find all of the garbage pages and add them to the list of free pages. This process is called garbage collection and imposes additional overhead and complexity on the system.

# Recovery Schemes

1. **write-ahead logging** (**WAL**) is a family of techniques for providing [atomicity](https://en.wikipedia.org/wiki/Atomic_(computer_science)) and [durability](https://en.wikipedia.org/wiki/Durability_(database_systems)) (two of the [ACID](https://en.wikipedia.org/wiki/ACID) properties) in [database systems](https://en.wikipedia.org/wiki/Database_system).
2. In a system using WAL, all modifications are written to a [log](https://en.wikipedia.org/wiki/Database_log) before they are applied. Usually both redo and undo information is stored in the log.
3. 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.
4. WAL allows updates of a database to be done [in-place](https://en.wikipedia.org/wiki/In-place_algorithm). Another way to implement atomic updates is with [shadow paging](https://en.wikipedia.org/wiki/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.
5. [ARIES](https://en.wikipedia.org/wiki/Algorithms_for_Recovery_and_Isolation_Exploiting_Semantics) is a popular algorithm in the WAL family.
6. [File systems](https://en.wikipedia.org/wiki/File_system) typically use a variant of WAL for at least file system [metadata](https://en.wikipedia.org/wiki/Metadata) called [journaling](https://en.wikipedia.org/wiki/Journaling_file_system).
7. WAL protects all permanent tables and all system tables but is not used to protect the Transient Journal (TJ), since TJ records are stored in the WAL log. WAL also is not used to protect spool or volatile tables.
8. The WAL log is maintained as a separate logical file system from the normal table area. Whole cylinders are allocated to the WAL log, and it has its own index structure.
9. The WAL log data is a sequence of WAL log records and includes the following:
10. **•**Redo records, used for updating disk blocks and insuring file system consistency during restarts.
11. **•**TJ records used for transaction rollback.

# 