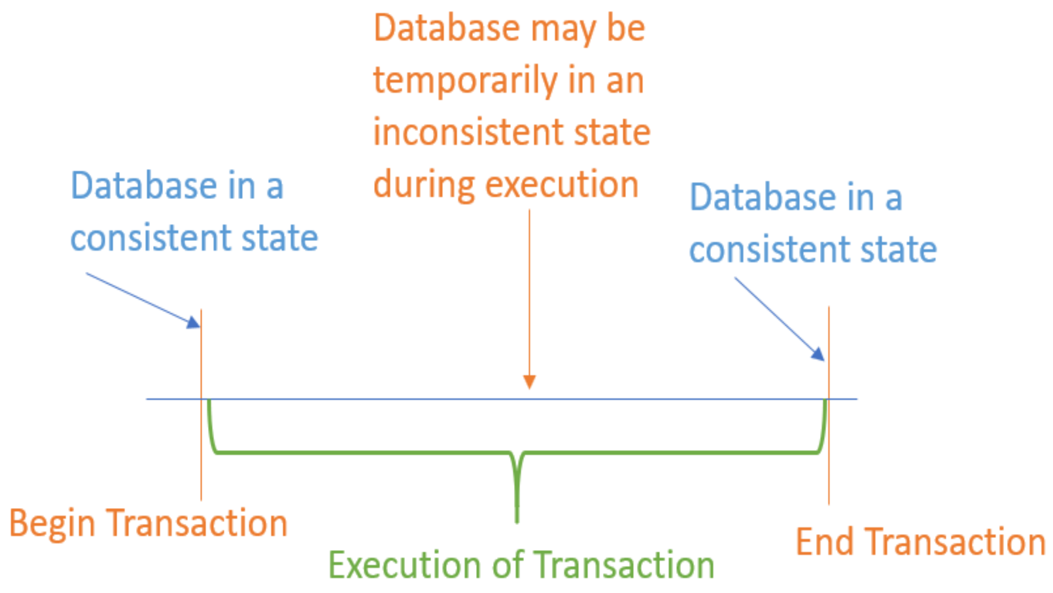
Database Transaction

A **DATABASE TRANSACTION** is a logical unit of processing in a DBMS which entails one or more database access operation. In a nutshell, database transactions represent real-world events of any enterprise.

All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.


Database may be temporarily in an inconsistent state during execution
Database in a consistent state
Database in a consistent state
Begin Transaction
Execution of Transaction
End Transaction

Facts about Database Transactions

- A transaction is a program unit whose execution may or may not change the contents of a database.
- The transaction is executed as a single unit
- If the database operations do not update the database but only retrieve data, this type of transaction is called a read-only transaction.
- A successful transaction can change the database from one CONSISTENT STATE to another
- DBMS transactions must be atomic, consistent, isolated and durable
- If the database were in an inconsistent state before a transaction, it would remain in the inconsistent state after the transaction.

ACID Properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

- **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left par
- tially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
- **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Concurrency Control

Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each another. Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical database, would have a mix of reading and WRITE operations and hence the concurrency is a challenge.

Concurrency control is used to address such conflicts which mostly occur with a multi-user system. It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases.

Therefore, concurrency control is a most important element for the proper functioning of a system where two or multiple database transactions that require access to the same data, are executed simultaneously.

Potential problems of Concurrency

Here, are some issues which you will likely to face while using the Concurrency Control method:

- **Lost Updates** occur when multiple transactions select the same row and update the row based on the value selected
- Uncommitted dependency issues occur when the second transaction selects a row which is updated by another transaction (**dirty read**)
- **Non-Repeatable Read** occurs when a second transaction is trying to access the same row several times and reads different data each time.
- **Incorrect Summary issue** occurs when one transaction takes summary over the value of all the instances of a repeated data-item, and second transaction update few instances of that specific data-item. In that situation, the resulting summary does not reflect a correct result.

Why use Concurrency method?

Reasons for using Concurrency control method is DBMS:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

Example

Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time.

However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose.

- Lock-Based Protocols
- Two Phase
- Timestamp-Based Protocols
- Validation-Based Protocols

Lock-based Protocols

A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks help synchronize access to the database items by concurrent transactions.

All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.

Binary Locks: A Binary lock on a data item can either locked or unlocked states.

Shared/exclusive: This type of locking mechanism separates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

1. Shared Lock (S):

A shared lock is also called a Read-only lock. With the shared lock, the data item can be shared between transactions. This is because you will never have permission to update data on the data item.

For example, consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

2. Exclusive Lock (X):

With the Exclusive Lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the 'write' operation.

For example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

3. Simplistic Lock Protocol

This type of lock-based protocols allows transactions to obtain a lock on every object before beginning operation. Transactions may unlock the data item after finishing the 'write' operation.

4. Pre-claiming Locking

Pre-claiming lock protocol helps to evaluate operations and create a list of required data items which are needed to initiate an execution process. In the situation when all locks are granted, the transaction executes. After that, all locks release when all of its operations are over.

DATABASE RECOVERY IN DBMS AND ITS TECHNIQUES:

There can be any case in database system like any computer system when database failure happens. So data stored in database should be available all the time whenever it is needed. So Database recovery means recovering the data when it get deleted, hacked or damaged accidentally. Atomicity is must whether is transaction is over or not it should reflect in the database permanently or it should not effect the database at all. So database recovery and database recovery techniques are must in DBMS. So database recovery techniques in DBMS are given below.

Crash recovery:

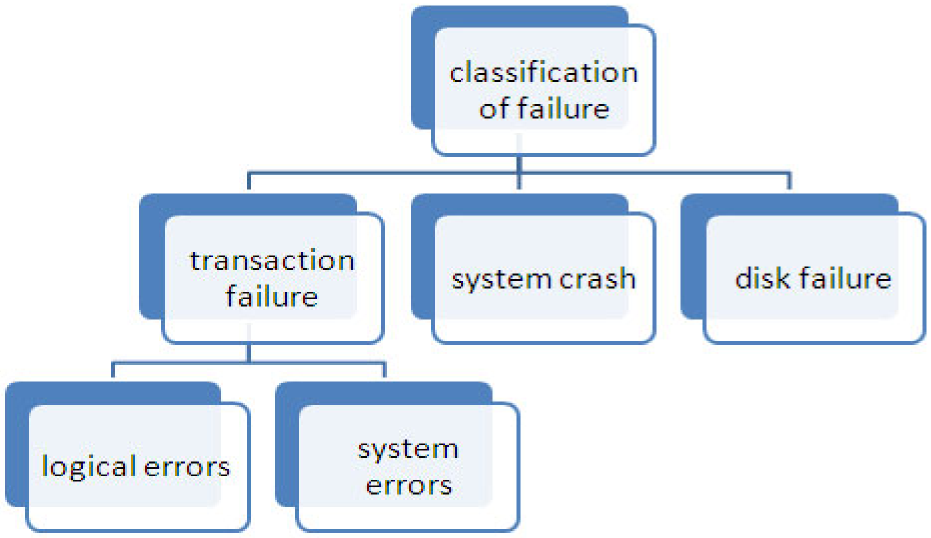
DBMS may be an extremely complicated system with many transactions being executed each second. The sturdiness and hardiness of software rely upon its complicated design and its underlying hardware and system package. If it fails or crashes amid transactions, it's expected that the system would follow some style of rule or techniques to recover lost knowledge.

DATABASE RECOVERY IN DBMS AND ITS TECHNIQUES

Classification of failure:

To see wherever the matter has occurred, we tend to generalize a failure into numerous classes, as follows:

- Transaction failure
- System crash
- Disk failure



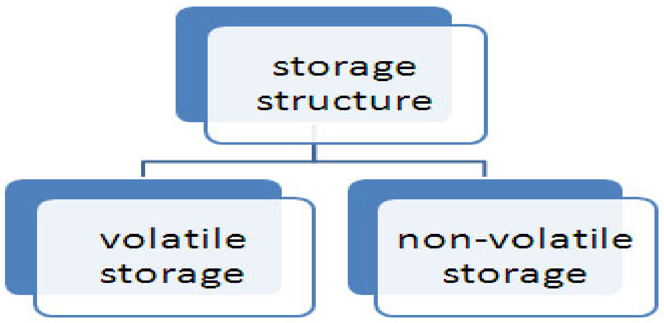
Types of Failure

1. **Transaction failure:** A transaction needs to abort once it fails to execute or once it reaches to any further extent from wherever it can't go to any extent further. This is often known as transaction failure wherever solely many transactions or processes are hurt. The reasons for transaction failure are:
    - Logical errors
    - System errors

1. **Logical errors:** Where a transaction cannot complete as a result of its code error or an internal error condition.
2. **System errors:** Wherever the information system itself terminates an energetic transaction as a result of the DBMS isn't able to execute it, or it's to prevent due to some system condition. to Illustrate, just in case of situation or resource inconvenience, the system aborts an active transaction.
3. **System crash:** There are issues − external to the system − that will cause the system to prevent abruptly and cause the system to crash. For instance, interruptions in power supply might cause the failure of underlying hardware or software package failure. Examples might include OS errors.
4. **Disk failure:** In early days of technology evolution, it had been a typical drawback wherever hard-disk drives or storage drives accustomed to failing oftentimes. Disk failures include the formation of dangerous sectors, unreachability to the disk, disk crash or the other failure, that destroys all or a section of disk storage.

Storage structure:

Classification of storage structure is as explained below:



Classification Of Storage

1. **Volatile storage:** As the name suggests, a memory board (volatile storage) cannot survive system crashes. Volatile storage devices are placed terribly near to the CPU; usually, they're embedded on the chipset itself. For instance, main memory and cache memory are samples of the memory board. They're quick however will store a solely little quantity of knowledge.
2. **Non-volatile storage:** These recollections are created to survive system crashes. they're immense in information storage capability, however slower in the accessibility. Examples could include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

Recovery and Atomicity:

When a system crashes, it should have many transactions being executed and numerous files opened for them to switch the information items. Transactions are a product of numerous operations that are atomic in nature. However consistent with ACID properties of a database, atomicity of transactions as an entire should be maintained, that is, either all the operations are executed or none.

When a database management system recovers from a crash, it ought to maintain the subsequent:

- It ought to check the states of all the transactions that were being executed.
- A transaction could also be within the middle of some operation; the database management system should make sure the atomicity of the transaction during this case.
- It ought to check whether or not the transaction is completed currently or it must be rolled back.
- No transactions would be allowed to go away from the database management system in an inconsistent state.

There are 2 forms of techniques, which may facilitate a database management system in recovering as well as maintaining the atomicity of a transaction:

- Maintaining the logs of every transaction, and writing them onto some stable storage before truly modifying the info.
- Maintaining shadow paging, wherever the changes are done on a volatile memory, and later, and the particular info is updated.

Log-based recovery Or Manual Recovery):

Log could be a sequence of records, which maintains the records of actions performed by dealing. It’s necessary that the logs area unit written before the particular modification and hold on a stable storage media, that is failsafe. Log-based recovery works as follows:

- The log file is unbroken on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log regarding it.