**ACID Properties in DBMS**

ACID refers to Atomicity, Consistency, Isolation and Durability.

A transaction is a logical unit of work that accesses and possibly modifies content of a database.

Atomicity in DBMS ensures that all operations within a transaction are completed all successfully or none at all. This guarantees data integrity by preventing partial updates.

Consistency ensures that a database remains in a valid state before and after a transaction. This means any transaction will bring the database from one valid state to another, maintaining all predefined rules.

Isolation in ACID properties ensures that transactions are executing independently without inference.

This guarantees that once a transaction is completed, its effects are permanently recorded. This ensures data remains intact even if the system crashes.

In Atomicity, there are two operations; abort: changes made are not visible and commit: changes are made visible.

For example, in a banking system transferring money from Account A to Account B, atomicity ensures that if debiting Account A succeeds but crediting Account B fails, the debit operation will be rolled back, preventing inconsistencies. This property simplifies error handling and contributes significantly to the database's reliability and robustness.

Inconsistency occurs when the data in the database is not correct. In the case of sending money, if the money is deducted from one account but not added to the next account the total amount of money in the database now becomes incorrect or inconsistent. If this is the case, a rollback is initiated to return the state of the database to the previous valid state.

For example, if a transaction involves transferring money between two accounts, the total balance in the system should remain consistent. If money is deducted from one account but not credited to the other, this results in an inconsistency. In such cases, the database triggers a rollback, rejecting the transaction and preserving its prior valid state.

Isolation makes it so that multiple transactions can happen simultaneously without one transaction influencing the other. Changes occurring in a transaction would not be visible unless the changes have been committed into the database. It makes sure the changes occur in the order that they were executed.

Consider two transactions:

Transaction A : Transfers $500 from Account X to Account Y.

Transaction B: Reads the balance of both Account X and Account Y.

Without isolation, Transaction B might read the intermediate state where $500 has been deducted from Account X but not yet added to Account Y, leading to incorrect results. Isolation ensures that Transaction B either sees the state before or after Transaction A, but not the intermediate state.

Durability ensures that after changes have been committed into the memory, they become permanent and are stored on a disk. If there is any crash in the system or any failure in the system, the changes committed still persist in the database and can be accessed without losing any info.

For example, in a banking system, if a transaction involves transferring money and the system crashes after the transaction is committed, durability ensures that the transfer has been completed and recorded in the database