**Acid properties in DBMS**

TRANSACTION

A transaction is an action or series of actions that are being performed which reads or updates the contents of the database. In order to maintain consistency in a database there are certain properties that all transactions should follow and possess before and after the transaction. The **four basic**  are in combination termed as ACID properties.

A - Atomicity

C - Consistency

I - Isolation

D - Durability

**Atomicity**

Atomicity is also known as the “all or nothing rule”.

By this, we mean that either the entire transaction takes place at once or doesn’t happen at all. There is no midway which means

transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

**Abort**: If a transaction aborts, changes made to the database are not  visible.  
**Commit**: If a transaction commits, changes made are visible.

**EXAMPLE**

Consider the following transaction from person A to person B. We want to transfer 1000 Rs from person A to person B’s account.

Before transaction

|  |  |
| --- | --- |
| **Person (A) = 4000** | **Person (B) = 3000** |

Transaction

|  |  |
| --- | --- |
| Person (A) | Person (B) |
| Read (A) | Read(B) |
| A: A-1000 | B: B+1000 |
| Write (A) | Write (B) |

After Transaction

|  |  |
| --- | --- |
| Person (A) = 3000 | Person(B) = 4000 |

If the amount has been deducted from A’s account but not added to B’s account (after write (A) but before write (B)) it would result  in an inconsistent database state. Therefore, the transaction must be executed entirely in order to ensure the correctness of the database state.

**Consistency**

The database must be consistent before and after the transaction.

Referring to the example above,

The total amount before and after the transaction must be maintained.   
Total before the transaction occurs=4000 + 3000 = 7000

Total after the transaction occurs =3000 + 4000 = 7000

Therefore, the database is **consistent**.

**Isolation**

The term isolation means separation. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

Transactions occur independently without interference (serial schedule and not parallel schedule).

Example

Let X = 50 and Y= 50.

Now, consider the following transaction T1 and T2.

|  |  |
| --- | --- |
| T1 | T2 |
| Read (X) | Read(X) |
| X: X\*100 | Read(Y) |
| Write(X) | Z: = X+Y |
| Read(Y) | Write(Z) |
| Y: = Y-50 |  |
| Write(Y) |  |

.

Suppose **T1** has been executed till **Read (Y)** and then **T2** starts. As a result, interleaving of operations takes place due to which T2 reads the correct value of **X** but the incorrect value of **Y** and sum computed by   
 **T2: X+Y = 500+50=550**

is thus not consistent with the sum at end of the transaction:   
 **X+Y=500+0=500**

.   
This will result in database inconsistency. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

**Durability**

In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database.

* When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
* The recovery subsystem of the DBMS has the responsibility of Durability property.