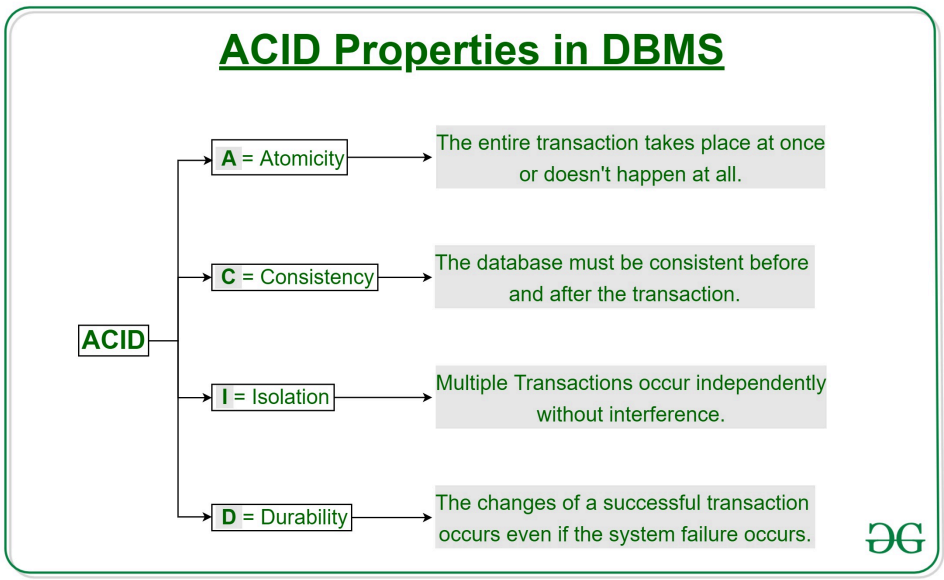
# ACID Properties in DBMS

Last Updated : 15 Apr, 2025

In the world of **Database Management Systems** (DBMS), transactions are fundamental operations that allow us to modify and retrieve data. However, to ensure the integrity of a database, it is important that these transactions are executed in a way that maintains consistency, correctness, and reliability. This is where the **ACID properties** come into play.

ACID stands for **Atomicity**, **Consistency**, **Isolation**, and **Durability**. These four key properties define how a transaction should be processed in a reliable and predictable manner, ensuring that the database remains consistent, even in cases of failures or concurrent accesses.




## What Are Transactions in DBMS?

A **transaction** in DBMS refers to a sequence of operations performed as a single unit of work. These operations may involve reading or writing data to the database. To maintain data integrity, DBMS ensures that each transaction adheres to the **ACID properties**. Think of a transaction

like an ATM withdrawal. When we withdraw money from our account, the transaction involves several steps:

- Checking your balance.
- Deducting the money from your account.
- Adding the money to the bank's record.

For the transaction to be successful, **all steps** must be completed. If any part of this process fails (e.g., if there's a system crash), the entire transaction should fail, and no data should be altered. This ensures the database remains in a **consistent** state.

## The Four ACID Properties

### 1. Atomicity: "All or Nothing"

**Atomicity** ensures that a transaction is **atomic**, it means that either the entire transaction completes fully or doesn't execute at all. There is no in-between state i.e. transactions do not occur partially. If a transaction has multiple operations, and one of them fails, the whole transaction is rolled back, leaving the database unchanged. This avoids partial updates that can lead to inconsistency.

- **Commit**: If the transaction is successful, the changes are permanently applied.
- **Abort/Rollback**: If the transaction fails, any changes made during the transaction are discarded.

**Example**: Consider the following transaction **T** consisting of **T1** and **T2** : Transfer of $100 from account **X** to account **Y** .

| Before: X : 500 | Y : 200 |
|---|---|
| Transaction T | |
| T1 | T2 |
| Read (X)<br>X: = X - 100<br>Write (X) | Read (Y)<br>Y: = Y + 100<br>Write (Y) |
| After: X : 400 | Y : 300 |

*Example*

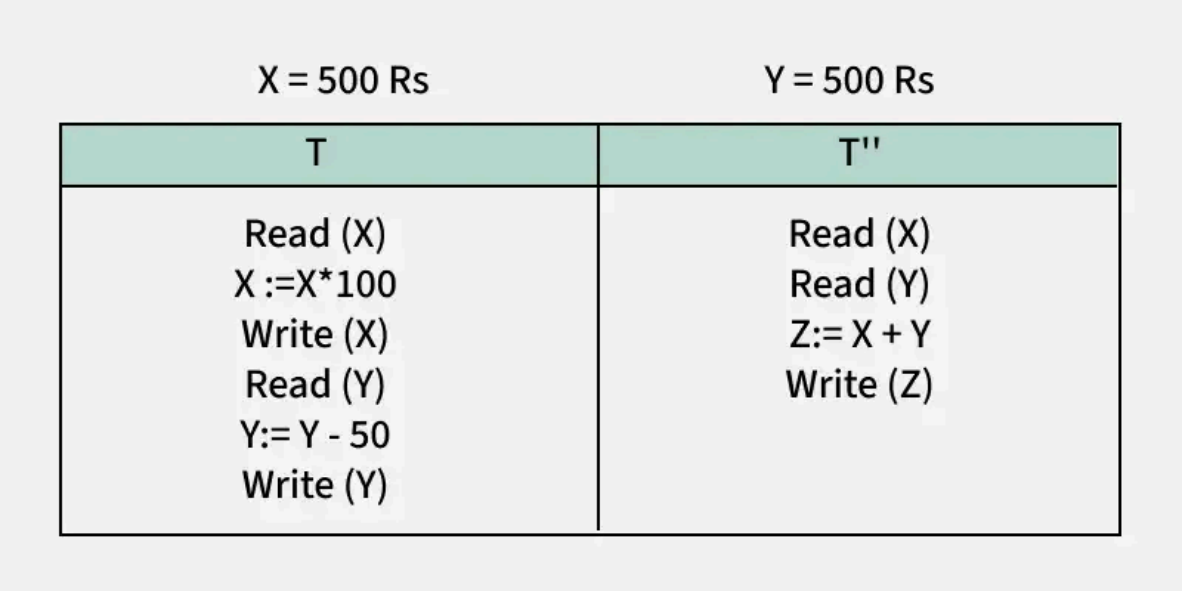
If the transaction fails after completion of T1 but before completion of T2 , the database would be left in an inconsistent state. With Atomicity, if any part of the transaction fails, the entire process is rolled back to its original state, and no partial changes are made.

## 2. Consistency: Maintaining Valid Data States

Consistency ensures that a database remains in a valid state before and after a transaction. It guarantees that any transaction will take the database from one consistent state to another, maintaining the rules and constraints defined for the data. In simple terms, a transaction should only take the database from one **valid** state to another. If a transaction violates any database rules or constraints, it should be rejected, ensuring that only consistent data exists after the transaction.

**Example**: Suppose the sum of all balances in a bank system should always be constant. Before a transfer, the total balance is **$700**. After the transaction, the total balance should remain $700. If the transaction fails in the middle (like updating one account but not the other), the system should maintain its consistency by rolling back the transaction

**Total before T occurs** = 500 + 200 = 700 .
**Total after T occurs** = 400 + 300 = 700 .



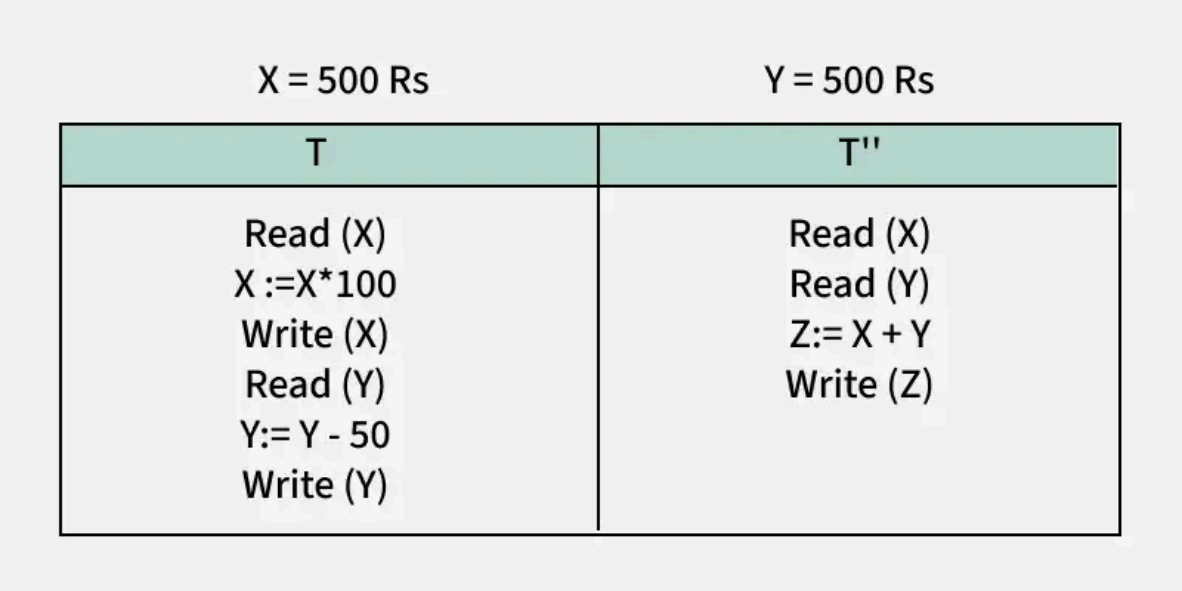

### 3. Isolation: Ensuring Concurrent Transactions Don't Interfere

This property ensures that **multiple transactions** can occur concurrently without leading to the **inconsistency** of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.

This property ensures that when multiple transactions run at the same time, the result will be the same as if they were run one after another in a specific order. This property prevents issues such as **dirty reads** (reading uncommitted data), **non-repeatable reads** (data changing between two reads in a transaction), and **phantom reads** (new rows appearing in a result set after the transaction starts).

**Example**: Let's consider two transactions:Consider two transactions **T** and **T''.**

- **X = 500, Y = 500**




**Transaction T:**

- **T** wants to transfer $50 from **X** to **Y**.
- **T** reads **Y** (value: 500), deducts $50 from **X** (new **X** = **450**), and adds $50 to **Y** (new **Y** = **550**).

**Transaction T'':**

- **T''** starts and reads **X** (value: 500) and **Y** (value: 500), then calculates the sum: **500** + **500** = **1000**.

But, by the time **T** finishes, **X** and **Y** have changed to **450** and **550** respectively, so the correct sum should be **450** + **550** = **1000**. **Isolation** ensures that **T''** should not see the old values of **X** and **Y** while **T** is still in progress. Both transactions should be independent, and **T''** should only see the final state after **T** commits. This prevents inconsistent data like the incorrect sum calculated by **T''**

## 4. Durability: Persisting Changes

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in **non-volatile memory**. In the event of a failure, the DBMS can recover the database to the state it was in after the last committed transaction, ensuring that no data is lost.

**Example**: After successfully transferring money from Account A to Account B, the changes are stored on disk. Even if there is a crash immediately after the commit, the transfer details will still be intact when the system recovers, ensuring durability.

## How ACID Properties Impact DBMS Design and Operation

The ACID properties, in totality, provide a mechanism to ensure the correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

### 1. Data Integrity and Consistency

ACID properties safeguard the **data integrity** of a DBMS by ensuring that transactions either complete successfully or leave no trace if interrupted. They prevent **partial updates** from corrupting the data and ensure that the database transitions only between valid states.

### 2. Concurrency Control

ACID properties provide a solid framework for **managing concurrent transactions**. Isolation ensures that transactions do not interfere with each other, preventing data anomalies such as lost updates, temporary inconsistency, and uncommitted data.

### 3. Recovery and Fault Tolerance

Durability ensures that even if a system crashes, the database can recover to a consistent state. Thanks to the **Atomicity** and **Durability** properties, if a transaction fails midway, the database remains in a consistent state.

| Property | Responsibility for maintaining properties |
| --- | --- |
| Atomicity | Transaction Manager |
| Consistency | Application programmer |
| Isolation | Concurrency Control Manager |
| Durability | Recovery |

## Advantages of ACID Properties in DBMS

1. **Data Consistency:** ACID properties ensure that the data remains consistent and accurate after any transaction execution.
2. **Data Integrity:** It maintains the integrity of the data by ensuring that any changes to the database are permanent and cannot be lost.
3. **Concurrency Control:** ACID properties help to manage multiple transactions occurring concurrently by preventing interference between them.
4. **Recovery:** ACID properties ensure that in case of any failure or crash, the system can recover the data up to the point of failure or crash.

## Disadvantages of ACID Properties in DBMS

1. **Performance Overhead**: ACID properties can introduce performance costs, especially when enforcing isolation between transactions or ensuring atomicity.
2. **Complexity**: Maintaining ACID properties in distributed systems (like microservices or cloud environments) can be complex and may require sophisticated solutions like distributed locking or transaction coordination.
3. **Scalability Issues**: ACID properties can pose scalability challenges, particularly in systems with high transaction volumes, where traditional relational databases may struggle under load.

## ACID in the Real World: Where Is It Used?

In modern applications, ensuring the **reliability and consistency** of data is crucial. ACID properties are fundamental in sectors like:

- **Banking**: Transactions involving money transfers, deposits, or withdrawals must maintain strict consistency and durability to prevent errors and fraud.
- **E-commerce**: Ensuring that inventory counts, orders, and customer details are handled correctly and consistently, even during high traffic, requires ACID compliance.
- **Healthcare**: Patient records, test results, and prescriptions must adhere to strict consistency, integrity, and security standards.

## Conclusion

The **ACID properties** in DBMS provide the backbone for maintaining data consistency, integrity, and reliability in the face of transaction failures, concurrent operations, and system crashes. In today's digital world, **ACID properties** ensure that database systems can handle complex transactions securely, reliably, and efficiently, which is why they remain a cornerstone of data management systems used in a variety of critical applications.

By understanding how ACID works, developers and **system administrators** can design better systems and make informed decisions about database management, especially when it comes to balancing consistency, performance, and scalability.



**Next Article**

Implementation of Locking in DBMS

## Similar Reads

**Base Properties in DBMS**

Pre-requisites: ACID Properties in DBMS The BASE properties of a database management system are a set of principles that guide the...