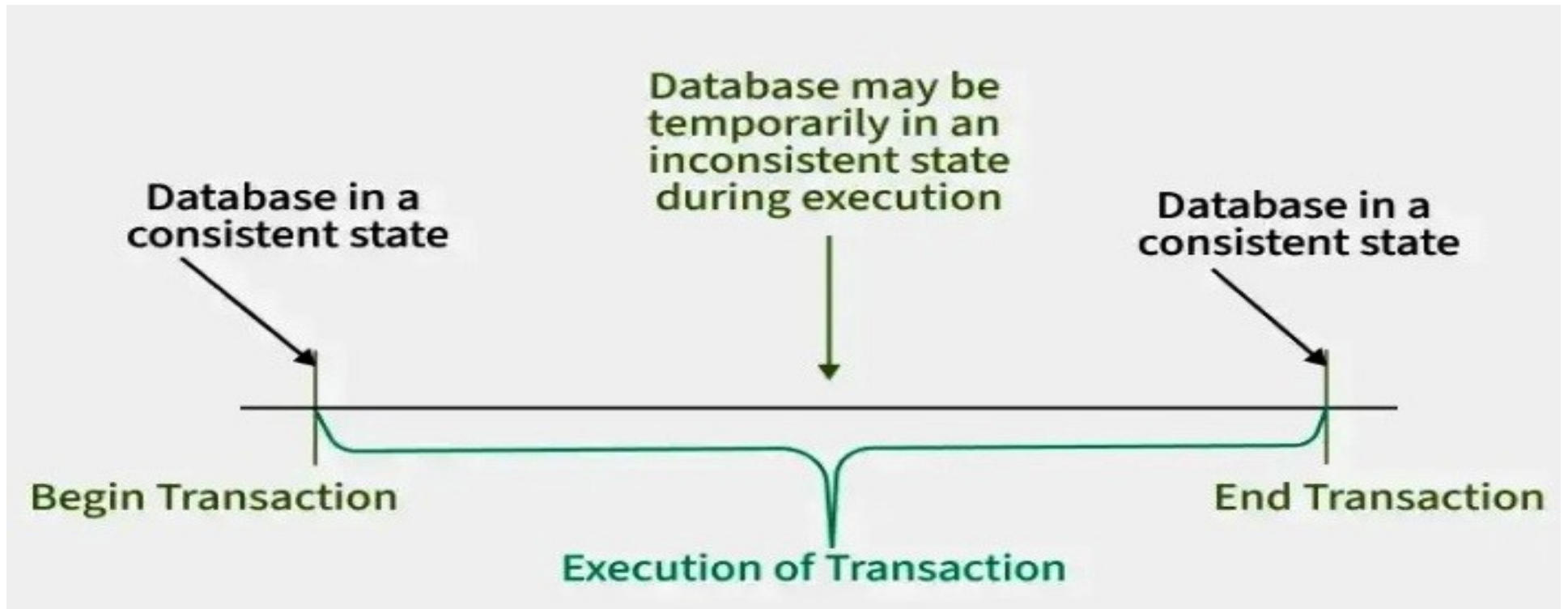


Transaction and its States in DBMS

Dr. Anish Soni

Introduction

- A **transaction** in a Database Management System (DBMS) is a sequence of one or more operations executed as a single logical unit of work.
- The primary purpose of transaction management is to ensure data integrity, consistency, and reliability, even in the event of system failures, network crashes, or concurrent user access
- A transaction ensures all operations are completed or none are applied (commit or rollback).



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.

Basic Transaction Operations

A transaction interacts with a database through four fundamental operations:

- **Read(X)**: Retrieves the value of a data item X from the database and stores it in the main memory buffer.
- **Write(X)**: Takes the updated value of X from the main memory buffer and writes it back to the database.
- **Commit**: Finalizes the transaction, making all its updates permanent in the database.
- **Rollback**: Undoes all operations performed by the transaction if an error occurs, restoring the database to its last consistent state.

Basic Transaction Operations

Example: Fund Transfer

Consider transferring Rs 500/- from Account A to Account B.

This represents a single transaction involving multiple steps:

1. Read(A)
2. $A := A - 500$
3. Write(A)
4. Read(B)
5. $B := B + 500$
6. Write(B)
7. Commit

If the system crashes at step 4, money is deducted from A but not added to B. The DBMS must rollback steps 1-3 to prevent data corruption

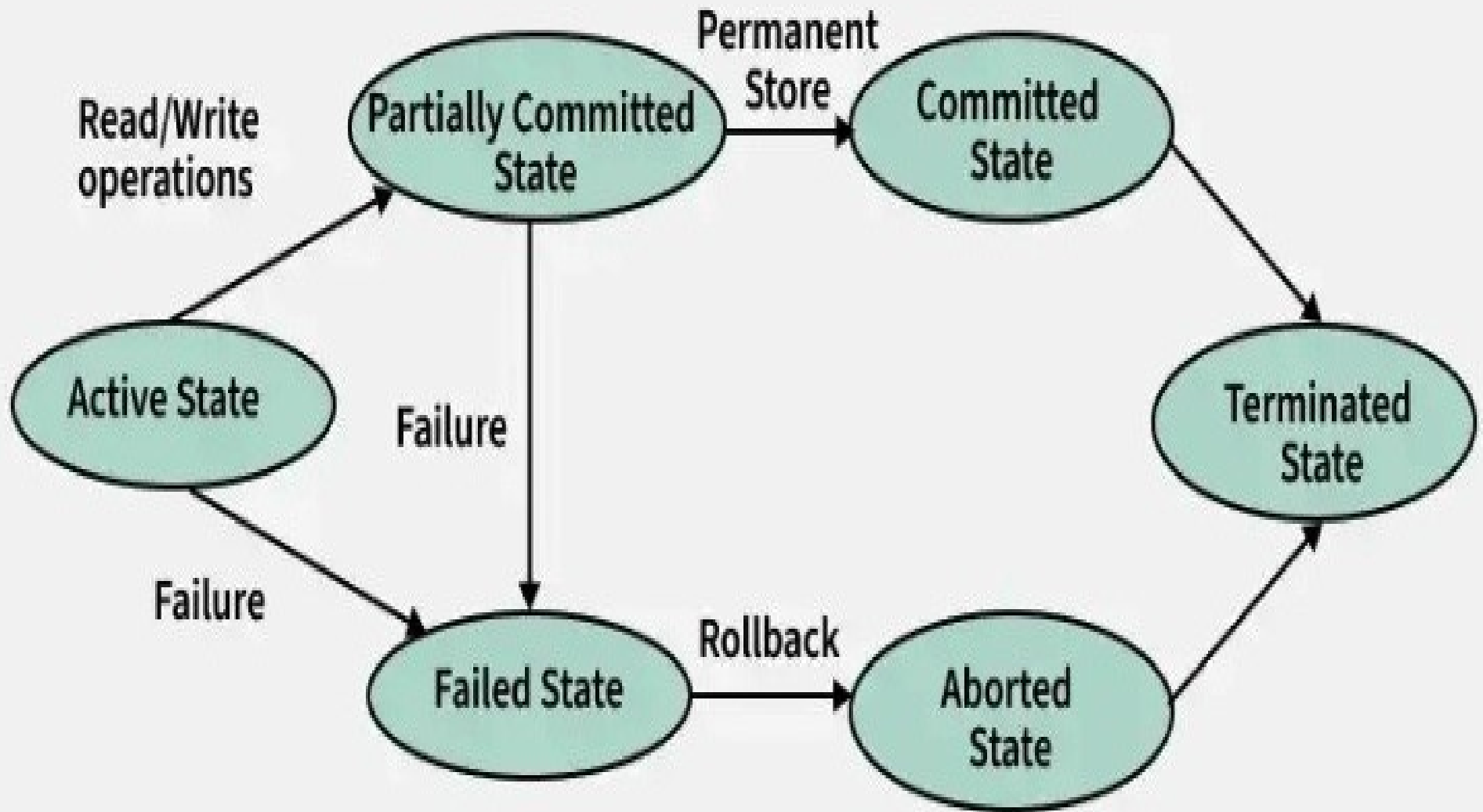
Transaction States in DBMS

Transaction states represent the different phases a transaction goes through during its lifecycle in a database system. These states help track the progress and outcome of a transaction to ensure data consistency and integrity.

These are the different types of Transaction States:

- Active State
- Partially Committed State
- Committed State
- Failed State
- Aborted State
- Terminated State

Transition States in DBMS



1. Active State

It is the first stage of any [transaction](#) when it has begun to execute. The execution of the transaction takes place in this state.

Operations such as insertion, deletion, or updation are performed during this state.

During this state, the data records are under manipulation and they are not saved to the database, rather they remain somewhere in a buffer in the main memory.

2. Partially Committed

The transaction has finished its final operation, but the changes are still not saved to the database.

After completing all read and write operations, the modifications are initially stored in main memory or a local buffer. If the changes are made permanent in the database then the state will change to "committed state" and in case of failure it will go to the "failed state".

3. Committed

This state of transaction is achieved when all the transaction-related operations have been executed successfully along with the Commit operation, i.e. data is saved into the database after the required manipulations in this state. This marks the successful completion of a transaction.

4. Failed State

If any of the transaction-related operations cause an error during the active or partially committed state, further execution of the transaction is stopped and it is brought into a failed state. Here, the database recovery system makes sure that the database is in a consistent state.

5. Aborted State

If a transaction reaches the failed state due to a failed check, the database recovery system will attempt to restore it to a consistent state. If recovery is not possible, the transaction is either rolled back or cancelled to ensure the database remains consistent.

In the aborted state, the DBMS recovery system performs one of two actions:

Kill the transaction: The system terminates the transaction to prevent it from affecting other operations.

Restart the transaction: After making necessary adjustments, the system reverts the transaction to an active state and attempts to continue its execution.

6. Terminated State

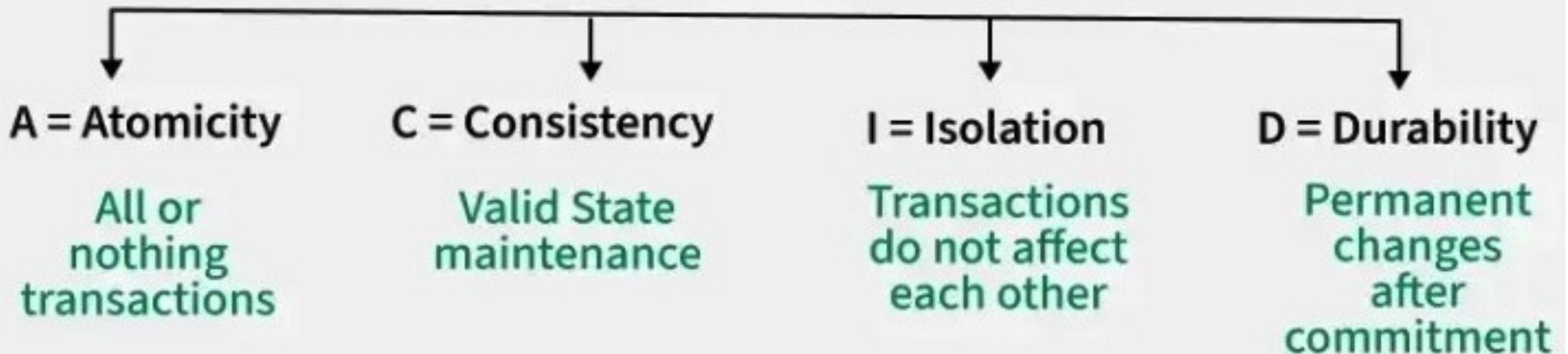
It refers to the final state of a transaction, indicating that it has completed its execution. Once a transaction reaches this state, it has either been successfully committed or aborted. In this state, no further actions are required from the transaction, as the database is now stable.

ACID PROPERTIES

- 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 even in the case of failures/errors. This is where the ACID properties come into play.

Acid Properties in DBMS

ACID



ACID PROPERTIES cont..

1. Atomicity

- Atomicity means a transaction is all-or-nothing either all its operations succeed, or none are applied. If any part fails, the entire transaction is rolled back to keep the database consistent.
- **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) X: = X - 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

Atomicity

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.

ACID PROPERTIES cont..

2. Consistency

- Consistency in transactions means that the database must remain in a valid state before and after a transaction.
- A valid state follows all defined rules, constraints, and relationships (like primary keys, foreign keys, etc.).
- If a transaction violates any of these rules, it is rolled back to prevent corrupt or invalid data.
- If a transaction deducts money from one account but doesn't add it to another (in a transfer), it violates consistency.
- **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 = $400 + 300 = 700$.

ACID PROPERTIES cont..

- **3. Isolation**
- Isolation ensures that transactions run independently without affecting each other. Changes made by one transaction are not visible to others until they are committed.
- It ensures that the result of concurrent transactions is the same as if they were run one after another, preventing issues like:
- **Dirty reads:** reading uncommitted data
- **Non-repeatable reads:** data changes between two reads

X = 500 Rs		Y = 500 Rs	
T		T''	
Read (X)		Read (X)	
X := X - 50		Read (Y)	
Write (X)		Z := X + Y	
Read (Y)		Write (Z)	
Y := Y + 50			
Write (Y)			

ACID PROPERTIES cont..

- **Example:** Consider two transactions T and T'.
- **X = 500, Y = 500**
- **Explanation:**
- **1. 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).
- **2. Transaction T':**
- T' starts and reads X (500) and Y (500).
- It calculates the sum: $500 + 500 = 1000$.
- Meanwhile, values of X and Y change to 450 and 550 respectively.
- So, the correct sum should be $450 + 550 = 1000$.
- Isolation ensures that T' does not read outdated values while another transaction (T) is still in progress.
- Transactions should be independent, and T' should access the final values only after T commits.
- This avoids inconsistent results, like the incorrect sum calculated by T'.

ACID PROPERTIES cont..

- **4. Durability:**
- Durability ensures that once a transaction is committed, its changes are permanently saved, even if the system fails. The data is stored in non-volatile memory, so the database can recover to its last committed state without losing data.
- **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.

How ACID Properties Impact DBMS Design and Operation

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