TRANSACTION:

Transaction refers to the small unit of any given program that consists of various low level tasks. A set of logically related operations is known as a transaction.

The main operation in the transaction are:

READ(A):

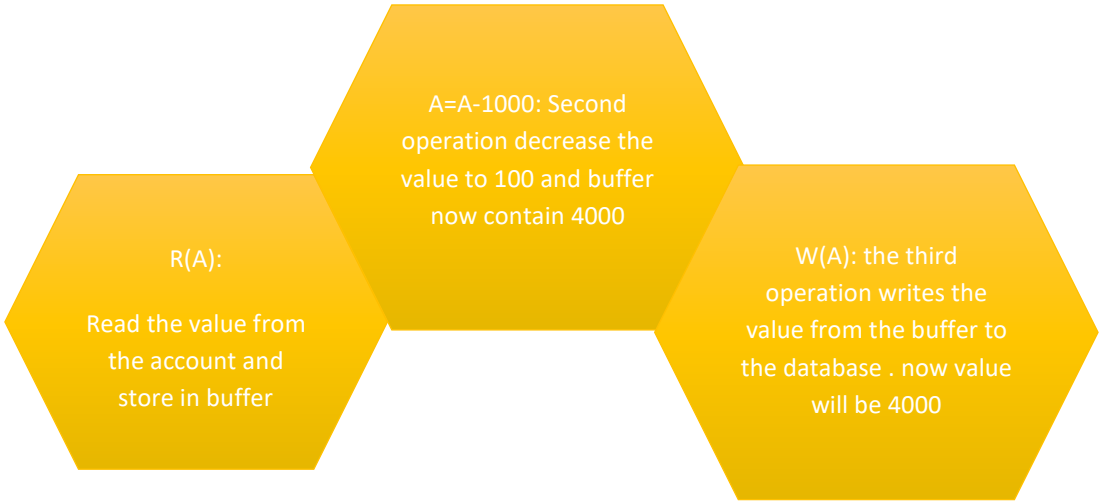
Read operations Read(A) or R(A) reads the value of A from the database and stores it in a buffer in the main memory.

Buffer memory is a temporary storage area in the main memory (RAM) that stores data transferring between two or more devices or between an application and a device.

WRITE (A):

Write operation Write(A) or W(A) writes the value back to the database from the buffer.

Let’s take a debit transaction from an account, before starting the transaction tha amount is 5000.



But it may also be possible that the transaction may fail after executing some of its operations. The failure can be because of hardware, software or power, etc. For example, if the debit transaction discussed above fails after executing operation 2, the value of A will

remain 5000 in the database which is not acceptable by the bank. To avoid this, Database has two important operations:

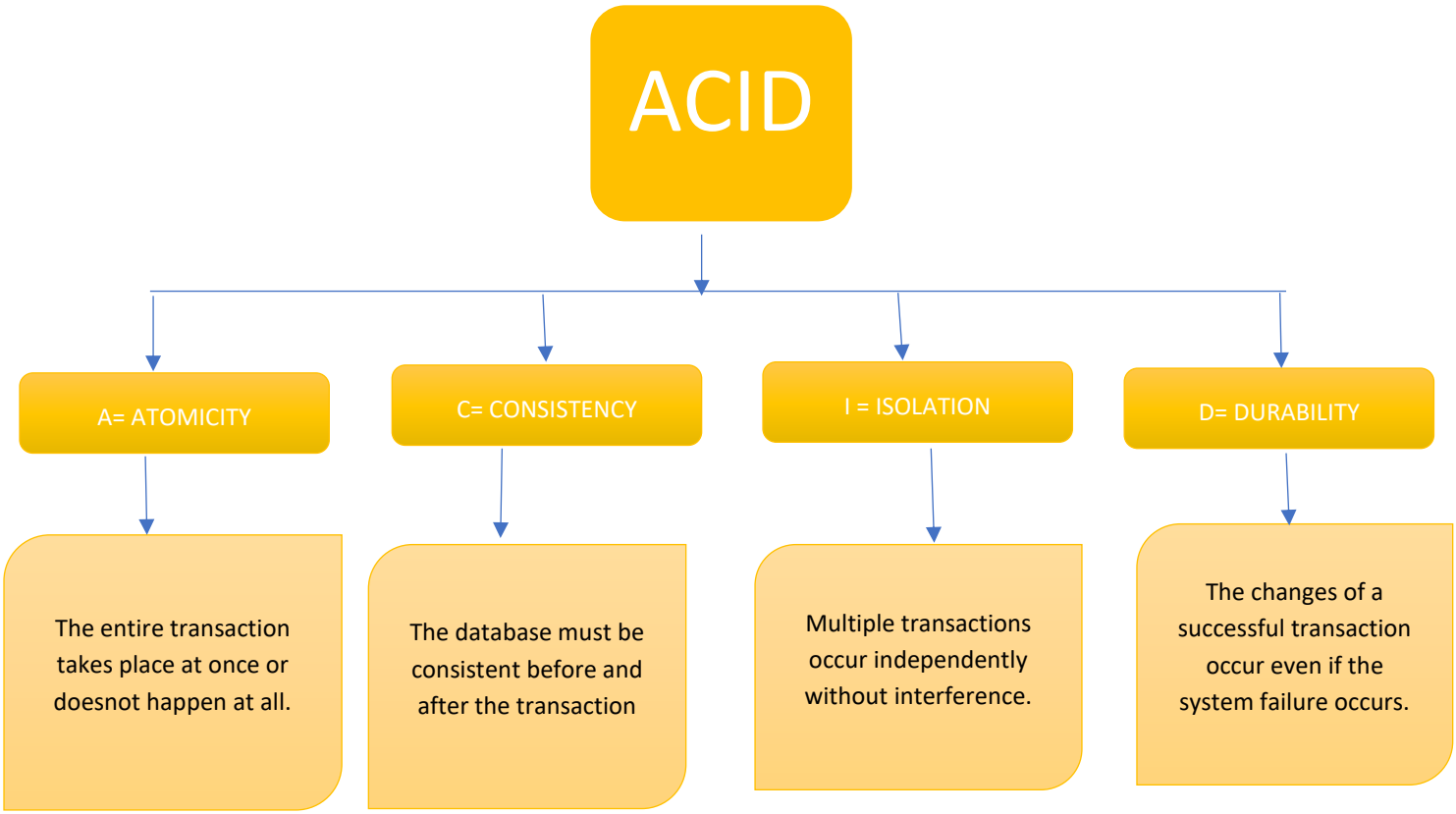
COMMIT:

After all instructions of a transaction are successfully executed, the changes made by a transaction are made permanent in the database.

ROLLBACK:

If a transaction is not able to execute all operations successfully, all the changes made by a transaction are undone.

PROPERTIES OF TRANSACTION:



ATOMICITY:

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

Atomicity is also known as the 'All or nothing rule'.

CONSISTENCY:

If operations of debit and credit transactions on the same account are executed concurrently, it may leave the database in an inconsistent state.

For Example, with T1 (debit of Rs. 1000 from A) and T2 (credit of 500 to A) executing concurrently, the database reaches an inconsistent state.

Let us assume the Account balance of A is Rs. 5000. T1 reads A(5000) and stores the value in its local buffer space. Then T2 reads A(5000) and also stores the value in its local buffer space.

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
	A=5000	R(A);	A=5000	A=5000
A=A-1000;	A=4000		A=5000	A=5000
	A=4000	A=A+500;	A=5500	
W(A);			A=5500	A=4000
		W(A);		A=5500

T1 performs A=A-1000 (5000-1000=4000) and 4000 is stored in T1 buffer space. Then T2 performs A=A+500 (5000+500=5500) and 5500 is stored in the T2 buffer space. T1 writes the value from its buffer back to the database.

A's value is updated to 4000 in the database and then T2 writes the value from its buffer back to the database. A's value is updated to 5500 which shows that the effect of the debit transaction is lost and the database has become inconsistent.

To maintain consistency of the database, we need concurrency control protocols which will be discussed in the next article. The operations of T1 and T2 with their buffers and database have been shown in table.

ISOLATION:

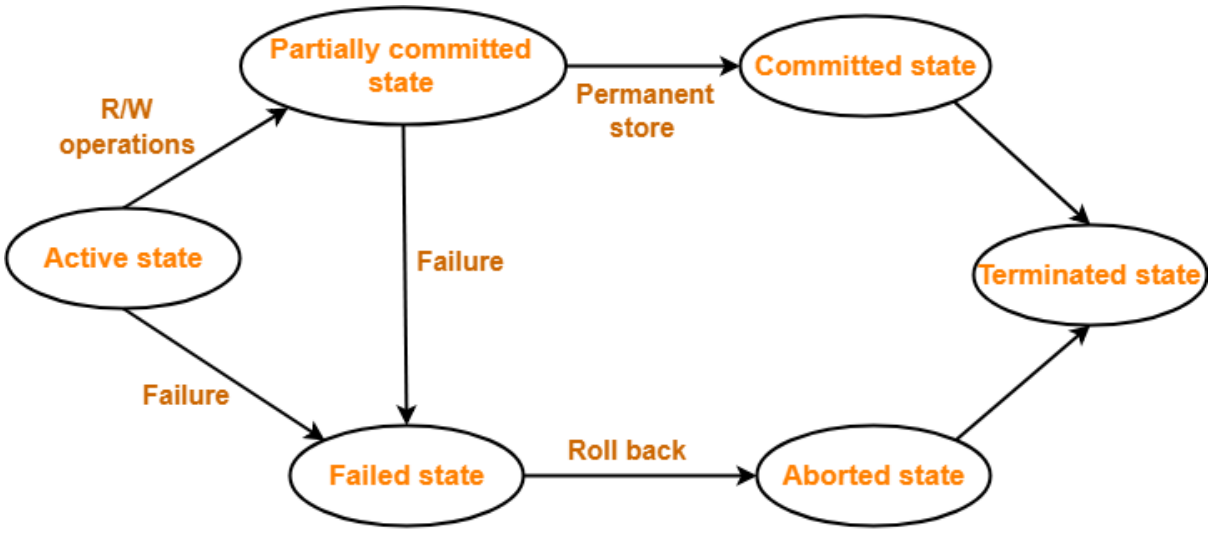
This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state.

The result of a transaction should not be visible to others before the transaction is committed. For example, let us assume that A's balance is Rs. 5000 and T1 debits Rs. 1000 from A. A's new balance will be 4000. If T2 credits Rs. 500 to A's new balance, A will become 4500, and after this T1 fails. Then we have to roll back T2 as well because it is using the value produced by T1. So transaction results are not made visible to other transactions before it commits.

DURABLE:

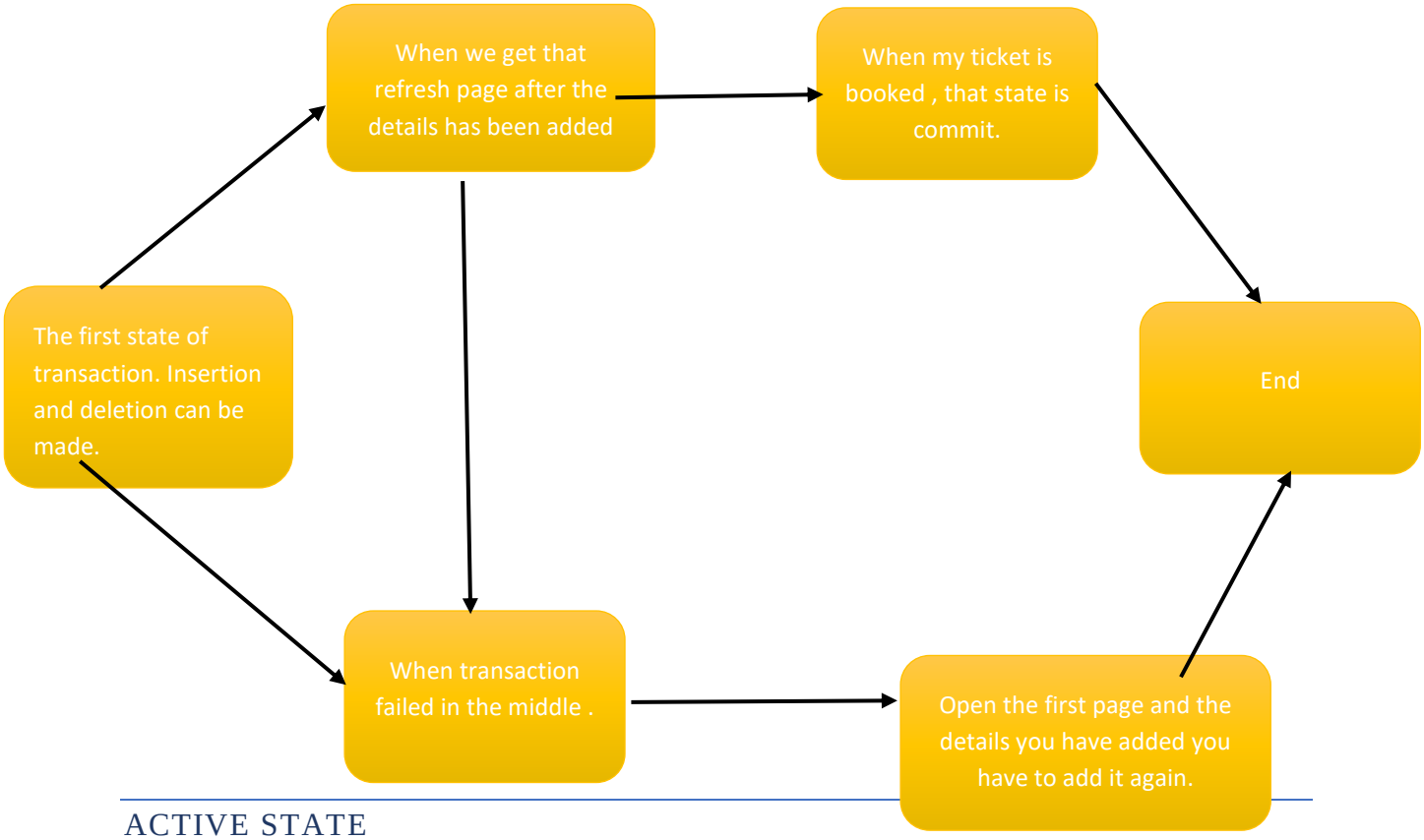
Once the database has committed a transaction, the changes made by the transaction should be permanent. e.g.; If a person has credited $500000 to his account, the bank can't say that the update has been lost. To avoid this problem, multiple copies of the database are stored at different locations.

STATES IN TRANSACTION:



Transaction States in DBMS

Lets take example of the transaction we do during the train ticket booking.



ACTIVE STATE

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

PARTIALLY COMMITTED

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

COMMITTED

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

FAILED STATE

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.

- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

ABORTED

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction