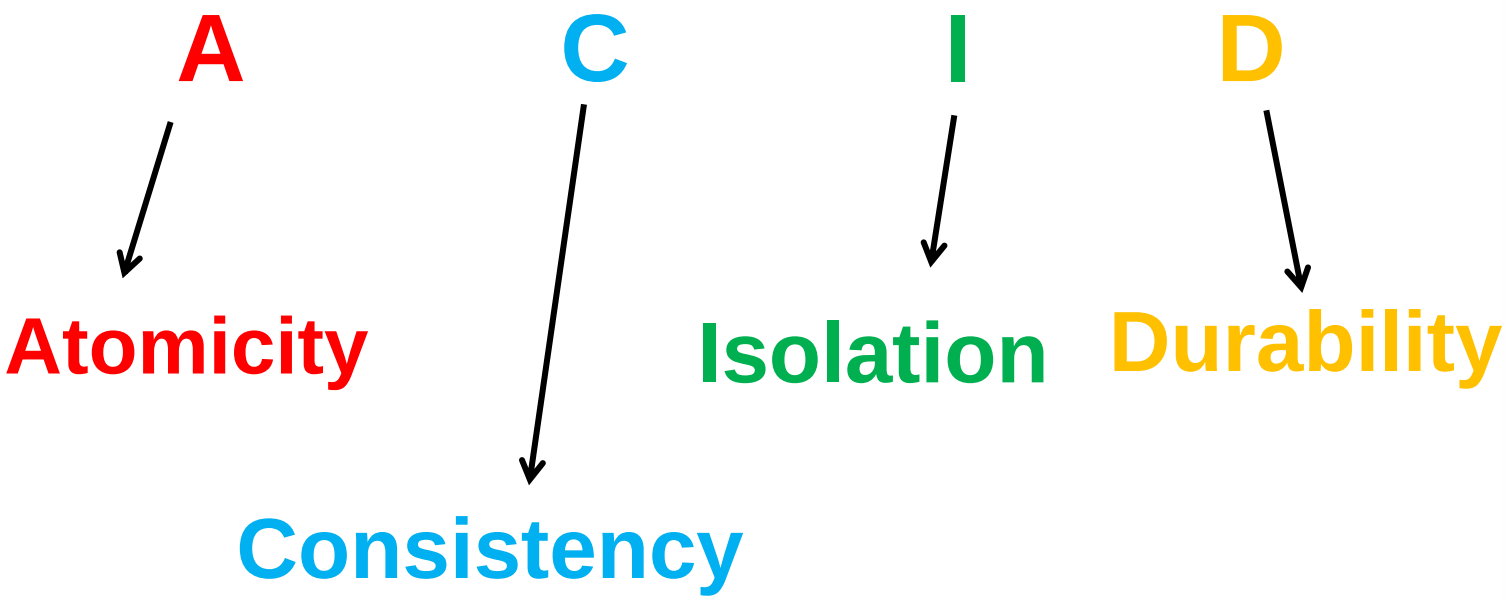
Transaction Processing

Lecture 1

What is a Transaction?

- **A transaction is a logical unit of database processing that must be completed in its entirety to ensure correctness**
- A transaction is a collection of operations that form a single logical unit of work which has to be executed completely
- Transaction is initiated by a user program written in a high-level language where it is delimited by statements of the form **begin transaction** and **end transaction**
- It consists of all operations within these statements

Properties of a Transaction



ACID properties – 1/2

Atomicity

Either all operations of the transaction are performed properly or none are

Consistency

Execution of a transaction must take from one consistent state to another

ACID properties – 2/2

Isolation

A transaction should appear as though it is being executed in isolation from other transactions

Durability

After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures

Operations

- Transactions access data using two operations
  - **read(X)** : Transfers data item X from database to a local buffer belonging to the transaction that executed the **read** operation
  - **write(X)** : Transfers the data item X from local buffer of the transaction that executed the **write** back to the database

Example for A C I D [1/7]

- Let T be a transaction that transfers Rs.500 from account A to B. This transaction is defined as follows

```
T :     read(A);
        A:=A-500;
        write(A);
        read(B);
        B:=B+500;
        write(B);
```

Example for A C I D [2/7]

- **Consistency :**
  - Consistency requirement is sum of A and B be unchanged by execution of transaction
  - If database is consistent before execution of transaction, the database must remain consistent after execution of transaction
  - Ensuring consistency is the responsibility of the **application programmer** who codes the transaction

Example for A C I D [3/7]

- **Atomicity**
  - Suppose A = Rs.1000 and B = Rs.2000
  - During execution of T, failure occurs that prevents T from completing its execution successfully (Power failure, hardware failures..)
  - Suppose that failure happened after write(A) operation but before write(B) operation
  - Here, values of accounts A = Rs. 500 and B = Rs. 2000
  - Rs. 500 is lost (as a result of failure)
  - Hence the sum A+B is no longer preserved

Example for A C I D [4/7]

- The database is in the **inconsistent state**
- The database system keeps track (on disk) of old values of any data on which a transaction performs a write, and if transaction does not complete its execution, the database system restores the old values to make it appear as though the transaction never executed
- Ensuring atomicity is responsibility of database system itself, handled by a component called **transaction-management component**

Example for A C I D [5/7]

- **Durability**
  - Once a transaction completes successfully, all the updates that is carried out on database persist, even if there is a system failure after the transaction completes execution
  - We can guarantee durability by ensuring that either
    - The updates carried out by the transaction have been written to disk before transaction completes
    - Information about updates carried out by the transaction and written to disk is sufficient to enable the database to reconstruct the updates when the database system is restarted after the failure
  - Ensuring durability is the responsibility of a s/w component of database system called **recovery-management component**

Example for A C I D [6/7]

- **Isolation**
  - Even if consistency and atomicity properties are ensured for each transaction, if several transactions are executed concurrently, their operations may interleave in some undesirable way, resulting in an inconsistent state
  - Eg. While transaction to transfer funds from A to B is executing, with deducted total written to A and increased total yet to be written to B, if a second concurrently running transaction reads A and B at this intermediate point and computes A+B, it will observe inconsistent value
  - Also, if second transaction then performs updates on A and B based on the inconsistent values that it read, the database may be left in inconsistent state even after both transaction have completed

Example for A C I D [7/7]

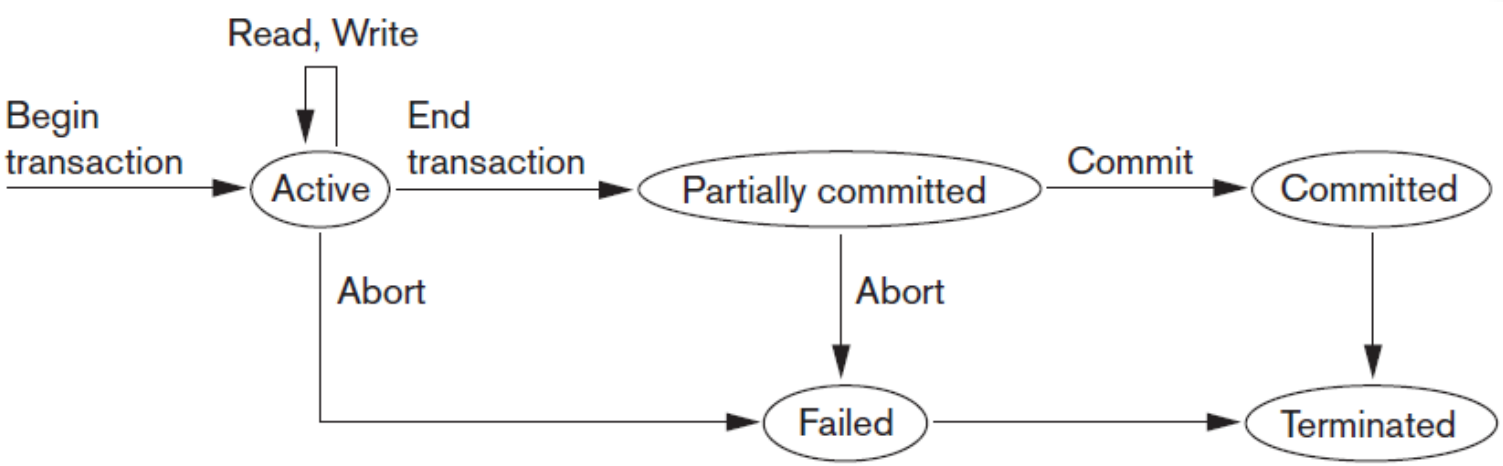
- To avoid this, execute transactions serially one after the other
- Isolation property ensures that the concurrent execution of transactions results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in some order
- This is the responsibility of **concurrency-control component**

Transaction States [1/5]

- Transaction must be in one of the following states
  - **Active**:- Initial state; transaction stays in this state while it is executing
  - **Partially committed**:- After the final statement has been executed
  - **Failed**:- After the discovery that normal execution can no longer proceed
  - **Aborted**:- After transaction has been rolled back and the database has been restored to its state prior to the start of the transaction
  - **Committed**:- After successful completion

Transaction States [2/5]

Figure : State diagram of transaction *



**** Fundamentals of Database Systems, 6th Edition, Elmasri & Navathe, Page 752***

Transaction States [3/5]

- A transaction starts in the **active** state
- When it finishes its final statement, it enters the **partially committed** state
- At this point, transaction has completed its execution but it is still possible that it may have to be aborted, since actual o/p may still be temporarily residing in memory, and thus h/w failure may disrupt the successful completion
- Database system then writes out enough information to disk that, even in the event of failure, the updates performed by transaction can be recreated when system restarts after the failure

Transaction States [4/5]

- When the last of this information is written out, transaction enters the **committed** state
- Transaction enters the **failed** state after the system determines that transaction can no longer proceed with its normal execution. Such a transaction is *rolled back*
- A transaction is said to have **terminated** if it has either committed or aborted

Transaction States [5/5]

- At this point, system has two options
  - It can **restart transaction**, but only if transaction was aborted as a result of some h/w or s/w error. A restarted transaction is said to be new transaction
  - It can **kill transaction**. It does so because some internal logical error that can be corrected only by rewriting application program, or because i/p was bad, or desired data was not found in database

Additional operations [1/4]

- For recovery purpose, system keeps track of when each transaction starts, terminates and commits or aborts
  - BEGIN_TRANSACTION
  - READ or WRITE
  - END_TRANSACTION
  - COMMIT_TRANSACTION
  - ROLLBACK (or ABORT)

Additional operations [2/4]

BEGIN_TRANSACTION

This marks the beginning of transaction execution.

READ or WRITE

These specify read or write operations on the database items that are executed as part of a transaction

Additional operations [3/4]

END_TRANSACTION

This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution.

However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability or for some other reason

Additional operations [4/4]

COMMIT_TRANSACTION

This signals a *successful end* of the transaction so that any changes (updates) executed by the transaction can be safely **committed** to the database and will not be undone

ROLLBACK (or ABORT)

This signals that the transaction has *ended unsuccessfully,* so that any changes or effects that the transaction may have applied to the database must be **undone**.