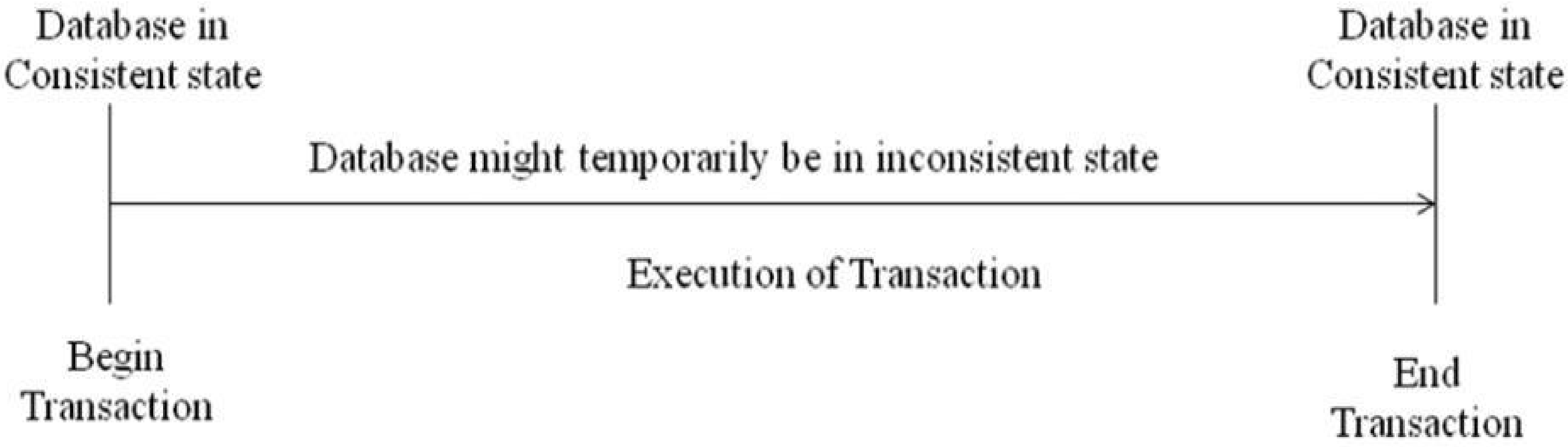
Transactions

Dr. Hrudaya Kumar Tripathy,
School of Computer Engineering, KIIT

Transaction Management Concept in DBMS

Transaction in DBMS

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- Transaction is a set of operations which are all logically related.
- A transaction is a logical unit of work that contains one or more SQL statements. The effects of all the SQL statements in a transaction can be either all committed or all rolled back.
- A transaction that changes the contents of the database must alter the database from one consistent database state to another.



ACID Properties of a Transaction

It is important to ensure that the database remains consistent before and after the transaction.

To ensure the consistency of database, certain properties are followed by all the transactions occurring in the system. These properties are called as ACID Properties of a transaction.

A = Atomicity

C = Consistency

I = Isolation

D = Durability

Atomicity:

- This property ensures that either the transaction occurs completely or it does not occur at all.
- In other words, it ensures that no transaction occurs partially. That is why, it is referred to as “All or nothing rule“.
- It is the responsibility of Transaction Control Manager to ensure atomicity of the transactions.

Consistency:

- This property ensures that integrity constraints are maintained.
- In other words, it ensures that the database remains consistent before and after the transaction.
- It is the responsibility of DBMS and application programmer to ensure consistency of the database.

Isolation:

- This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
- During execution, each transaction feels as if it is getting executed alone in the system.
- A transaction does not realize that there are other transactions as well getting executed parallely.
- Changes made by a transaction becomes visible to other transactions only after they are written in the memory.
- It is the responsibility of concurrency control manager to ensure isolation for all the transactions.

Durability:

- This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk.
- It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.
- It is the responsibility of recovery manager to ensure durability in the database.

Demonstration of ACID Properties

Let T_1 be a transaction that transfers $100 from account A to account B

T_1
Read(A);
A:=A-100;
Write(A);
Read(B);
B:=B+100;
Write(B);

Atomicity: The database system keeps track of the old values of any data on which a transaction performs a write, and if the transaction does not complete its execution, the database system restores the old values to make it appear as though the transaction never executed.

Consistency: Sum of A and B be unchanged by the execution of the transaction.

Isolation: The database is temporarily inconsistent while the transaction to transfer funds from account A to B is executing. The solutions are either execute transactions serially or concurrent execution of transactions provides significant performance benefits such as increased throughputs.

Durability: Once a transaction completes successfully, all the updates that is carried out on the database persist, even if there is a system failure after the transaction completes execution.

Operations in Transaction:

Read Operation-

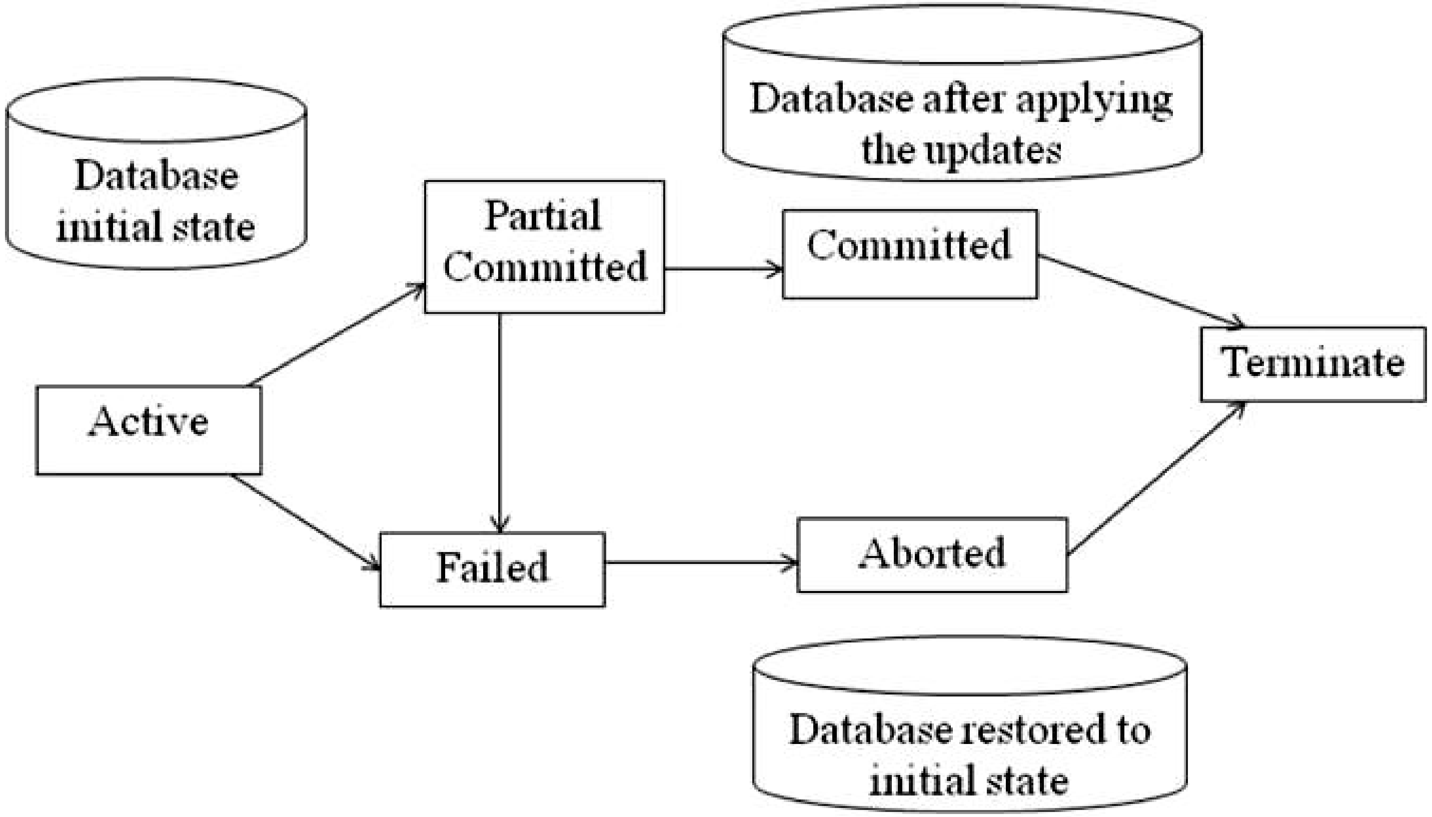
- Read operation reads the data from the database and then stores it in the buffer in main memory.
- For example- **Read(A)** instruction will read the value of A from the database and will store it in the buffer in main memory.

Write Operation-

- Write operation writes the updated data value back to the database from the buffer.
- For example- **Write(A)** will write the updated value of A from the buffer to the database.

Transaction States:

- A transaction goes through many different states throughout its life cycle.
- These states are called as transaction states.
- Transaction states are as follows-
  - **Active state**
  - **Partially committed state**
  - **Committed state**
  - **Failed state**
  - **Aborted state**
  - **Terminated state**



Transaction States Diagram

Transaction States

Active state: This state is the initial state of a transaction. The transaction stays in this state while it is executing.

Partial Committed state: A transaction is partial committed after its final statement has been executed. A transaction enters this state immediately before the commit statement.

Failed state: A transaction enters the failed state after the discovery that normal execution can no longer proceed.

Aborted state: A transaction is aborted after it has been rolled back and the database is restored to its prior state before the transaction.

Committed state: Committed state occurs after successful completion of the transaction.

Terminate: Transaction is either committed or aborted.

When a transaction enters the aborted state, the system has two options:

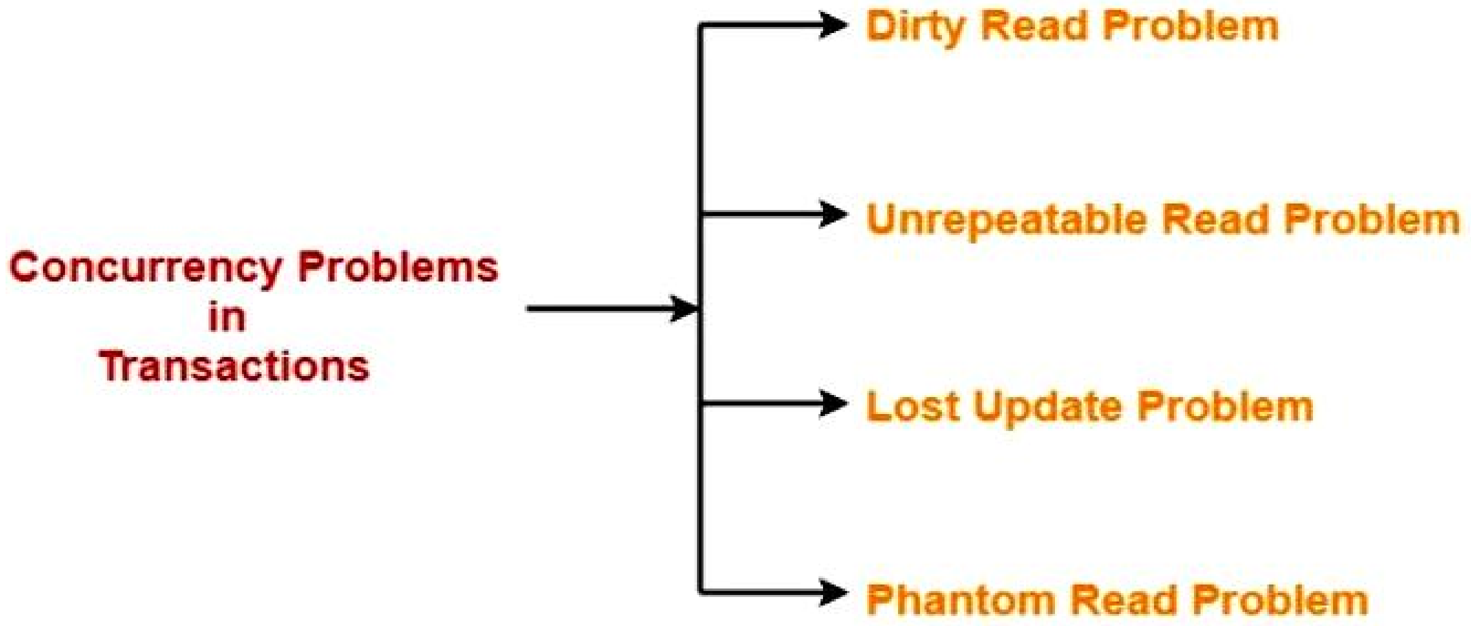
- Restart the transaction: If the transaction was aborted as a result of a hardware failure or some software error (other than logical error), it can be restarted.
- Kill the transaction: If the application program that initiated the transaction has some logical error.

Concurrency Problems in DBMS

- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems. Such problems are called as concurrency problems.
- Concurrent execution of transactions means executing more than one transaction at the same time.
- In the serial execution, one transaction can start executing only after the completion of the previous. The advantages of using concurrent execution of transactions are:

  Improved throughput and resource utilization Reduced waiting time

- The database system must control the interaction among the concurrent transactions to prevent them from destroying the consistency of the database. It does this through a variety of mechanisms called concurrency control schemes



Dirty Read Problem:

Reading the data written by an uncommitted transaction is called as dirty read.

This read is called as dirty read because-

- There is always a chance that the uncommitted transaction might roll back later.
- Thus, uncommitted transaction might make other transactions read a value that does not even exist.
- This leads to inconsistency of the database.

NOTE-

- Dirty read does not lead to inconsistency always.
- It becomes problematic only when the uncommitted transaction fails and roll backs later due to some reason.

Here,

1. T1 reads the value of A.
2. T1 updates the value of A in the buffer.
3. T2 reads the value of A from the buffer.
4. T2 writes the updated the value of A.
5. T2 commits.
6. T1 fails in later stages and rolls back.

Transaction T1	Transaction T2
R (A)	
W (A)	
	R (A) // Dirty Read
	W (A)
	Commit
Failure	

In this example,

- T2 reads the dirty value of A written by the uncommitted transaction T1.
- T1 fails in later stages and roll backs.
- Thus, the value that T2 read now stands to be incorrect.
- Therefore, database becomes inconsistent.

Unrepeatable Read Problem:

Here,

1. T1 reads the value of X (= 10 say).
2. T2 reads the value of X (= 10).
3. T1 updates the value of X (from 10 to 15 say) in the buffer.
4. T2 again reads the value of X (but = 15).

In this example,

- T2 gets to read a different value of X in its second reading.
- T2 wonders how the value of X got changed because according to it, it is running in isolation.

This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

Transaction T1	Transaction T2	
R (X)		
	R (X)	
W (X)		
	R (X)	// Unrepeated Read

Lost Update Problem:

This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Transaction T1	Transaction T2
R (A)	
W (A)	
	W (A)
	Commit
Commit	

Here,

1. T1 reads the value of A (= 10 say).
2. T2 updates the value to A (= 15 say) in the buffer.
3. T2 does blind write A = 25 (write without read) in the buffer.
4. T2 commits.
5. When T1 commits, it writes A = 25 in the database.

In this example,

- T1 writes the over written value of X in the database.
- Thus, update from T1 gets lost.

NOTE-

- This problem occurs whenever there is a write-write conflict.
- In write-write conflict, there are two writes one by each transaction on the same data item without any read in the middle.

Phantom Read Problem:

This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

Transaction T1	Transaction T2
R (X)	
	R (X)
Delete (X)	
	Read (X)

Here,

1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

- T2 finds that there does not exist any variable X when it tries reading X again.
- T2 wonders who deleted the variable X because according to it, it is running in isolation.

Avoiding Concurrency Problems-

- ✓ **To ensure consistency of the database, it is very important to prevent the occurrence of above problems.**

- ✓ **Concurrency Control Protocols help to prevent the occurrence of above problems and maintain the consistency of the database.**