International Islamic University Chittagong (IIUC)

Department of Computer and Communication Engineering

Program: **B.sc (Eng.)**
Course Code: **CCE–3505**
Type: **Theory**
Segment: **Final Term-Segment 7**

Baizid MD Ashadzzaman
Designation: **Adjunct Lecturer**
Email: **baizid.md.ashadzzaman@gmail.com**
Phone: **8801862420119**

Course Contents:

Contents	Note

Transaction in Database management System

Transaction:

A transaction is a set of logically related operations. For example, you are transferring money from your bank account to your friend's account, the set of operations would be like this:

Simple Transaction Example

1. Read your account balance
2. Deduct the amount from your balance
3. Write the remaining balance to your account
4. Read your friend's account balance
5. Add the amount to his account balance
6. Write the new updated balance to his account

This whole set of operations can be called a transaction.

In DBMS, we write the above 6 steps transaction like this:

Lets say your account is A and your friend's account is B, you are transferring 10000 from A to B, the steps of the transaction are:

1. R(A);
2. A = A - 10000;
3. W(A);
4. R(B);
5. B = B + 10000;
6. W(B); In the above transaction R refers to the Read operation and W refers to the write operation.

ACID properties of database transactions

The ACID properties are a set of four properties that guarantee the reliability and consistency of database transactions. Here's a brief description of each with an example:

Atomicity: This property ensures that either all the operations within a transaction are completed successfully or none of them are. If any part of the transaction fails, the entire transaction is rolled back, leaving the database in its original state.

Example: Suppose you're transferring money from one bank account to another. The transaction involves debiting the amount from one account and crediting it to another. If the debit operation succeeds but the credit operation fails (perhaps due to a network issue), atomicity ensures that the debit operation is undone, and the original amount remains in the source account.

Consistency: This property ensures that the database remains in a consistent state before and after the transaction. It means that transactions must preserve all the rules and constraints defined on the database, such as referential integrity constraints, datatype constraints, etc.

Example: Consider a scenario where you have a database constraint that enforces that all email addresses in a user table must be unique. If a transaction attempts to insert anew user with an email address that already exists in the database, the consistency property ensures that the transaction fails, and the database remains in a consistent state.

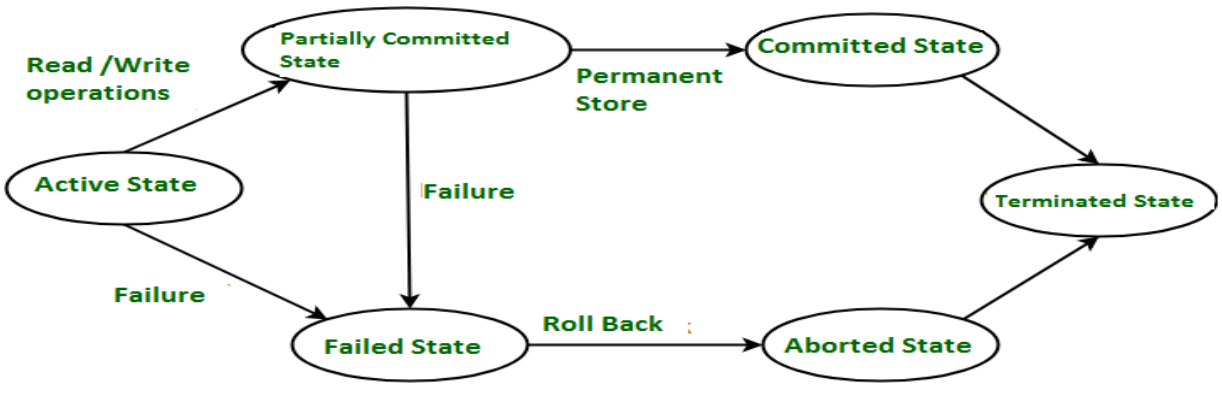
Isolation: This property ensures that the concurrent execution of transactions produces a result that is equivalent to some serial execution of those transactions. In other words, transactions should operate independently of each other, and the intermediate states should be hidden from other transactions until a transaction is committed.

Example: Suppose two transactions are simultaneously updating the same bank account balance. Isolation ensures that each transaction can't see the intermediate state of the other. It prevents scenarios like one transaction reading a partially updated value from another transaction, ensuring data integrity.

Durability: This property guarantees that once a transaction is committed, its changes are permanently saved, even in the event of a system failure. These changes should persist even if the system crashes or restarts.

Example: If you make a purchase online and the transaction is successfully completed (i.e., the payment is deducted from your account), the durability property ensures that the record of this transaction is safely stored in the database. Even if the server crashes immediately after your purchase, the transaction details will still be available and won't be lost.

DBMS Transaction States



Transaction States in DBMS

Active State

When the instructions of the transaction are running then the transaction is in active state. If all the 'read and write' operations are performed without any error then it goes to the "partially committed state"; if any instruction fails, it goes to the "failed state".

Partially Committed

After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the DataBase then the state will change to "committed state" and in case of failure it will go to the "failed state".

Failed State

When any instruction of the transaction fails, it goes to the "failed state" or if failure occurs in making a permanent change of data on Data Base.

Aborted State

After having any type of failure the transaction goes from "failed state" to "aborted state" and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.

Committed State

It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the "terminated state".

Terminated State

If there isn't any roll-back or the transaction comes from the "committed state", then the system is consistent and ready for new transaction and the old transaction is terminated.

Types of Schedules

In DBMS, Schedule is a process of lining the transactions and executing them one by one. When there are multiple transactions that are running in a concurrent manner and the order of operation is needed to be set so that the operations do not overlap each other.

Serial Schedules:

Schedules in which the transactions are executed non-interleaved, i.e., a serial schedule is one in which no transaction starts until a running transaction has ended are called serial schedules. i.e., In Serial schedule, a transaction is executed completely before starting the execution of another transaction. In other words, you can say that in serial schedule, a transaction does not start execution until the currently running transaction finished execution. This type of execution of transaction is also known as non interleaved execution. The example we have seen above is the serial schedule.

Example: Consider the following schedule involving two transactions T_1 and T_2.

T_1	T_2
R(A)	
W(A)	
R(B)	
	W(B)
	R(A)
	R(B)

Non-Serial Schedule:

This is a type of Scheduling where the operations of multiple transactions are interleaved. The transactions are executed in a non-serial manner.In the non-serial schedule, the other transaction proceeds without waiting for the previous transaction to complete. The Non-Serial Schedule can be divided further into Serializable and Non-Serializable.

Serializable:
This is used to maintain the consistency of the database. It is mainly used in the Non-Serial scheduling to verify whether the scheduling will lead to any inconsistency or not. The non-serial schedule is said to be in a serializable schedule only when it is equivalent to the serial schedules, for an n number of transactions. A serializable schedule helps in improving both resource utilization and CPU throughput. These are of two types:

Conflict Serializable:

A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. Two operations are said to be conflicting if all conditions satisfy:

- They belong to different transactions
- They operate on the same data item
- At Least one of them is a write operation

View Serializable:

A Schedule is called view serializable if it is view equal to a serial schedule (no overlapping transactions). A conflict schedule is a view serializable but if the serializability contains blind writes, then the view serializable does not conflict serializable.

Non-Serializable:
The non-serializable schedule is divided into two types, Recoverable and Non-recoverable Schedule.

Recoverable Schedule:
A schedule is recoverable if it allows for the recovery of the database to a consistent state after a transaction failure. In other words, if some transaction Tj is reading value updated or written by some other transaction Ti, then the commit of Tj must occur after the commit of Ti.

Example – Consider the following schedule involving two transactions T1 and T2.

T1	T2
R(A)	
W(A)	
	W(A)
	R(A)
commit	
	commit

This is a recoverable schedule since T1 commits before T2, that makes the value read by T2 correct.

Non-Recoverable Schedule:

A non-recoverable schedule means when there is a system failure, we may not be able to recover to a consistent database state.

Example: Consider the following schedule involving two transactions T1 and T2.

T1	T2
R(A)	
W(A)	
	W(A)
	R(A)
	commit
abort	

T2 read the value of A written by T1, and committed. T1 later aborted, therefore the value read by T2 is wrong, but since T2 committed, this schedule is non-recoverable.