**Transaction Management in DBMS**

**Database Transaction** is a logical unit of processing in a DBMS which entails one or more database access operation. In a nutshell, database transactions represent real-world events of any enterprise.

All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction in DBMS. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.

**Database Transactions**

* A transaction is a program unit whose execution may or may not change the contents of a database.
* The transaction concept in DBMS is executed as a single unit.
* If the database operations do not update the database but only retrieve data, this type of transaction is called a read-only transaction.
* A successful transaction can change the database from one CONSISTENT STATE to another
* DBMS transactions must be atomic, consistent, isolated and durable
* If the database were in an inconsistent state before a transaction, it would remain in the inconsistent state after the transaction.

**Example:** Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

**X's Account**

Open\_Account(X)

Old\_Balance = X.balance

New\_Balance = Old\_Balance - 800

X.balance = New\_Balance

Close\_Account(X)

**Y's Account**

Open\_Account(Y)

Old\_Balance = Y.balance

New\_Balance = Old\_Balance + 800

Y.balance = New\_Balance

Close\_Account(Y)

## Operations of Transaction:

Following are the main operations of transaction:

**Read(X):** Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

**Write(X):** Write operation is used to write the value back to the database from the buffer.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

**For example:** If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

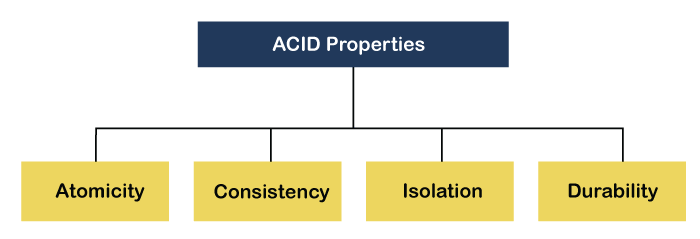
To solve this problem, we have two important operations:

**Commit:** It is used to save the work done permanently.

**Rollback:** It is used to undo the work done.

## ACID Properties

The expansion of the term ACID defines for:



**1) Atomicity:** The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

**Example:** If Remo has account A having $30 in his account from which he wishes to send $10 to Sheero's account, which is B. In account B, a sum of $ 100 is already present. When $10 will be transferred to account B, the sum will become $110. Now, there will be two operations that will take place. One is the amount of $10 that Remo wants to transfer will be debited from his account A, and the same amount will get credited to account B, i.e., into Sheero's account. Now, what happens - the first operation of debit executes successfully, but the credit operation, however, fails. Thus, in Remo's account A, the value becomes $20, and to that of Sheero's account, it remains $100 as it was previously present.

**2) Consistency:** The word **consistency** means that the value should remain preserved always. In [DBMS](https://www.javatpoint.com/dbms-tutorial), the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

The total amount before and after the transaction must be maintained.   
Total **before T** occurs = **500 + 200 = 700**.   
Total **after T occurs** = **400 + 300 = 700**.   
Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

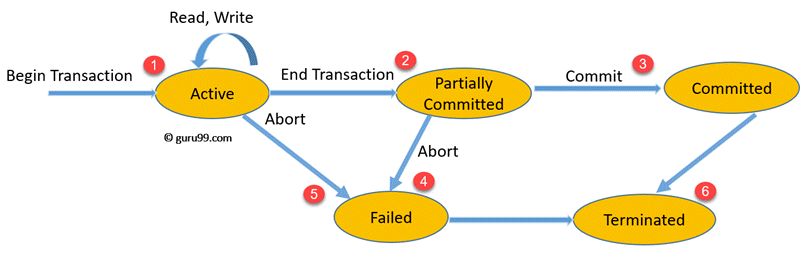
3) **Isolation**   
This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

4) **Durability:**   
This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

## States of Transactions

The various states of a transaction concept in DBMS are listed below:

|  |  |
| --- | --- |
| **State** | **Transaction types** |
| Active State | A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed. |
| Partially Committed | A transaction goes into the partially committed state after the end of a transaction. |
| Committed State | When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently. |
| Failed State | A transaction considers failed when any one of the checks fails or if the transaction is aborted while it is in the active state. |
| Terminated State | State of transaction reaches terminated state when certain transactions which are leaving the system can’t be restarted. |

State Transition Diagram for a Database Transaction

Let’s study a [state transition diagram](https://www.guru99.com/state-transition-testing.html) that highlights how a transaction moves between these various states.

1. Once a transaction states execution, it becomes active. It can issue READ or WRITE operation.
2. Once the READ and WRITE operations complete, the transactions becomes partially committed state.
3. Next, some recovery protocols need to ensure that a system failure will not result in an inability to record changes in the transaction permanently. If this check is a success, the transaction commits and enters into the committed state.
4. If the check is a fail, the transaction goes to the Failed state.
5. If the transaction is aborted while it’s in the active state, it goes to the failed state. The transaction should be rolled back to undo the effect of its write operations on the database.
6. The terminated state refers to the transaction leav ing the system.

**Serializability in DBMS**

Serializability is the concept in a transaction that helps to identify which non-serial schedule is correct and will maintain the database consistency. It relates to the isolation property of transaction in the database.

Serializability is the concurrency scheme where the execution of concurrent transactions is equivalent to the transactions which execute serially.

### **Serializable Schedule**

A serial schedule is always a serializable schedule because any transaction only starts its execution when another transaction has already completed its execution. However, a non-serial schedule of transactions needs to be checked for Serializability.

**Note: If a schedule of concurrent ‘n' transactions can be converted into an equivalent serial schedule. Then we can say that the schedule is serializable. And this property is known as serializability.**

### **Testing of Serializability**

To test the serializability of a schedule, we can use the serialization graph.

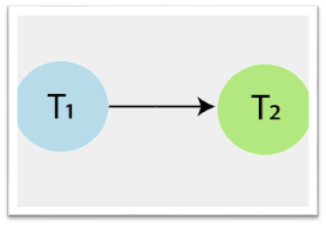
Suppose, a schedule S. For schedule S, construct a graph called as a precedence graph. It has a pair G = (V, E), where E consists of a set of edges, and V consists of a set of vertices. The set of vertices contain all the transactions participating in the S schedule. The set of edges contains all edges Ti ->Tj for which one of the following three conditions satisfy:

1. Create a node Ti ? Tj if Ti transaction executes write (Q) before Tj transaction executes read (Q).
2. Create a node Ti ? Tj if Ti transaction executes read (Q) before Tj transaction executes write (Q).
3. Create a node Ti ? Tj if Ti transaction executes write (Q) before Tj transaction executes write (Q).

**Schedule S:**

|  |  |  |
| --- | --- | --- |
| **Time** | **Transaction T1** | **Transaction T2** |
| t1 | Read(A) |  |
| t2 | A=A+50 |  |
| t3 | Write(A) |  |
| t4 |  | Read(A) |
| t5 |  | A+A+100 |
| t6 |  | Write(A) |

### **Precedence graph of Schedule S**



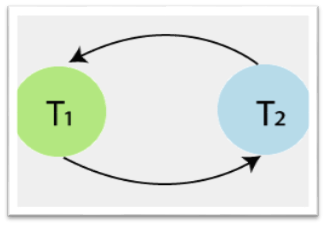
In above precedence graph of schedule S, contains two vertices T1 and T2, and a single edge T1? T2, because all the instructions of T1 are executed before the first instruction of T2 is executed.

If a precedence graph for any schedule contains a cycle, then that schedule is non-serializable. If the precedence graph has no cycle, then the schedule is serializable.  
So, schedule S is serializable (i.e., serial schedule) because the precedence graph has no cycle.

**Schedule S1:**

|  |  |  |
| --- | --- | --- |
| **Time** | **Transaction T1** | **Transaction T2** |
| t1 | Read(A) |  |
| t2 |  | Read(A) |
| t3 |  | Write(A) |
| t4 | A=A+50 |  |
| t5 | Write(A) |  |

**Precedence graph of Schedule S1**



In above precedence graph of schedule S1, contains two vertices T1 and T2, and edges T1? T2 and T2? T1. In this Schedule S1, operations of T1 and T2 transaction are present in an interleaved manner.  
The precedence graph contains a cycle, that’s why schedule S1 is non-serializable.

### **Types of Serializability**

1. Conflict Serializability
2. View Serializability