Transaction in DBMS

In **Database Management Systems (DBMS)**, a transaction is a fundamental concept representing a set of logically related operations executed as a **single unit**. **Transactions** are essential for handling user requests to access and modify database contents, ensuring the database remains consistent and reliable despite various operations and potential interruptions.

In this article, we will discuss what a transaction means, various operations of transactions, transaction states, and properties of transactions in DBMS.

What does a Transaction mean in DBMS?

- **Transaction** in **Database Management Systems** (**DBMS**) can be defined as a set of logically related operations.
- It is the result of a request made by the user to access the contents of the database and perform operations on it.
- It consists of various operations and has various states in its completion journey.
- It also has some specific properties that must be followed to keep the database consistent.

Operations of Transaction

A user can make different types of requests to access and modify the contents of a database. So, we have different types of operations relating to a transaction. They are discussed as follows:

i) Read(X)

- A **read operation** is used to read the value of **X** from the database and store it in a buffer in the main memory for further actions such as displaying that value.
- Such an operation is performed when a user wishes just to see any content of the database and not make any changes to it. For example, when a user wants to check **his/her** account's balance, a read operation would be performed on user's account balance from the database.

ii) Write(X)

- A write operation is used to write the value to the database from the buffer in the main memory. For a write operation to be performed, first a read operation is performed to bring its value in buffer, and then some changes are made to it, **e.g.** some set of arithmetic operations are performed on it according to the user's request, then to store the modified value back in the database, a write operation is performed.
- **For example**, when a user requests to withdraw some money from his account, his account balance is fetched from the database using a read operation, then the amount to be deducted from the account is subtracted from this value, and then the obtained value is stored back in the database using a write operation.

iii) Commit

- This operation in transactions is used to maintain integrity in the database. Due to some failure of power, hardware, or software, etc., a transaction might get interrupted before all its operations are completed. This may cause ambiguity in the database, i.e. it might get inconsistent before and after the transaction.

- To ensure that further operations of any other transaction are performed only after work of the current transaction is done, a commit operation is performed to the changes made by a transaction permanently to the database.

iv) Rollback

- This operation is performed to bring the database to the last saved state when any transaction is interrupted in between due to any power, hardware, or software failure.
- In simple words, it can be said that a rollback operation does undo the operations of transactions that were performed before its interruption to achieve a safe state of the database and avoid any kind of ambiguity or inconsistency.

Transaction Schedules

When multiple transaction requests are made at the same time, we need to decide their order of execution. Thus, a transaction schedule can be defined as a chronological order of execution of multiple transactions.

There are broadly two types of transaction schedules discussed as follows:

i) Serial Schedule

- In this kind of schedule, when multiple transactions are to be executed, they are executed serially, i.e. at one time only one transaction is executed while others wait for the execution of the current transaction to be completed. This ensures consistency in the database as transactions do not execute simultaneously.
- But, it increases the waiting time of the transactions in the queue, which in turn lowers the throughput of the system, i.e. number of transactions executed per time.
- To improve the throughput of the system, another kind of schedule are used which has some more strict rules which help the database to remain consistent even when transactions execute simultaneously.

ii) Non-Serial Schedule

- To reduce the waiting time of transactions in the waiting queue and improve the system efficiency, we use nonserial schedules which allow multiple transactions to start before a transaction is completely executed. This may sometimes result in inconsistency and errors in database operation.
- So, these errors are handled with specific algorithms to maintain the consistency of the database and improve **CPU** throughput as well.
- Non-serial schedules are also sometimes referred to as parallel schedules, as transactions execute in parallel in these kinds of schedules.

Serializable

- Serializability in DBMS is the property of a nonserial schedule that determines whether it would maintain the database consistency or not.
- The nonserial schedule which ensures that the database would be consistent after the transactions are executed in the order determined by that schedule is said to be Serializable Schedules.

- The serial schedules always maintain database consistency as a transaction starts only when the execution of the other transaction has been completed under it.
- Thus, serial schedules are always serializable.
- A transaction is a series of operations, so various states occur in its completion journey. They are discussed as follows:

i) Active

- It is the first stage of any transaction when it has begun to execute. The execution of the transaction takes place in this state.
- Operations such as insertion, deletion, or updation are performed during this state.
- During this state, the data records are under manipulation and they are not saved to the database, rather they remain somewhere in a buffer in the main memory.

ii) Partially Committed

- This state of transaction is achieved when it has completed most of the operations and is executing its final operation.
- It can be a signal to the commit operation, as after the final operation of the transaction completes its execution, the data has to be saved to the database through the commit operation.
- If some kind of error occurs during this state, the transaction goes into a failed state, else it goes into the Committed state.

iii) Commited

This state of transaction is achieved when all the transaction-related operations have been executed successfully along with the Commit operation, i.e. data is saved into the database after the required manipulations in this state. This marks the successful completion of a transaction.

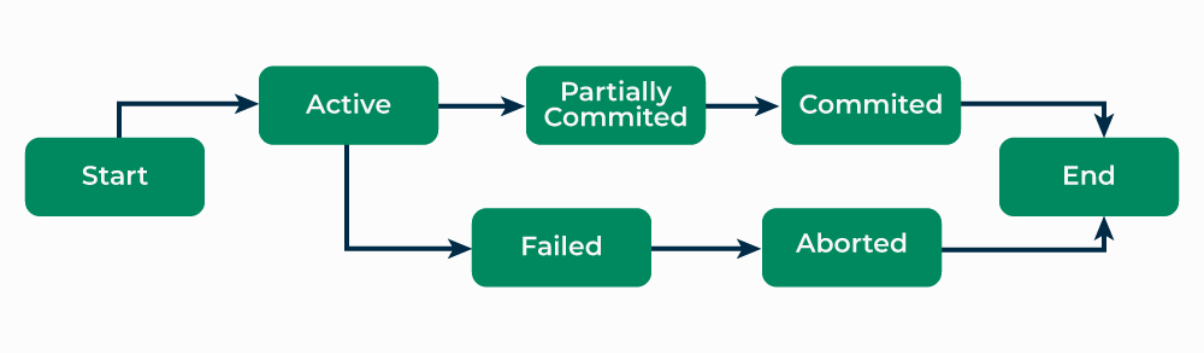
iv) Failed

- If any of the transaction-related operations cause an error during the active or partially committed state, further execution of the transaction is stopped and it is brought into a failed state. Here, the database recovery system makes sure that the database is in a consistent state.

v) Aborted

If the error is not resolved in the failed state, then the transaction is aborted and a rollback operation is performed to bring database to the the last saved consistent state. When the transaction is aborted, the database recovery module either restarts the transaction or kills it.

The illustration below shows the various states that a transaction may encounter in its completion journey.



Transaction in DBMS

Properties of Transaction

- As transactions deal with accessing and modifying the contents of the database, they must have some basic properties which help maintain the consistency and integrity of the database before and after the transaction. Transactions follow 4 properties, namely, **Atomicity, Consistency, Isolation, and Durability**.
- Generally, these are referred to as **ACID** properties of transactions in DBMS. ACID is the acronym used for transaction properties. A brief description of each property of the transaction is as follows.

i) Atomicity

- This property ensures that either all operations of a transaction are executed or it is aborted. In any case, a transaction can never be completed partially.
- Each transaction is treated as a single unit (like an atom). Atomicity is achieved through commit and rollback operations, i.e. changes are made to the database only if all operations related to a transaction are completed, and if it gets interrupted, any changes made are rolled back using rollback operation to bring the database to its last saved state.

ii) Consistency

- This property of a transaction keeps the database consistent before and after a transaction is completed.
- Execution of any transaction must ensure that after its execution, the database is either in its prior stable state or a new stable state.
- In other words, the result of a transaction should be the transformation of a database from one consistent state to another consistent state.
- Consistency, here means, that the changes made in the database are a result of logical operations only which the user desired to perform and there is not any ambiguity.

iii) Isolation

- This property states that two transactions must not interfere with each other, i.e. if some data is used by a transaction for its execution, then any other transaction can not concurrently access that data until the first transaction has completed.
- It ensures that the integrity of the database is maintained and we don’t get any ambiguous values. Thus, any two transactions are isolated from each other.
- This property is enforced by the concurrency control subsystem of DBMS.

iv) Durability

- This property ensures that the changes made to the database after a transaction is completely executed, are durable.
- It indicates that permanent changes are made by the successful execution of a transaction.
- In the event of any system failures or crashes, the consistent state achieved after the completion of a transaction remains intact. The recovery subsystem of DBMS is responsible for enforcing this property.

Conclusion

Transactions in DBMS are pivotal in maintaining the integrity and consistency of the database through a series of well-defined operations and states. From the initial execution of operations to handling errors and ensuring consistency, transactions must adhere to the ACID properties—Atomicity, Consistency, Isolation, and Durability. These properties guarantee that transactions are processed reliably and that the database remains stable even in the face of failures or concurrent operations.