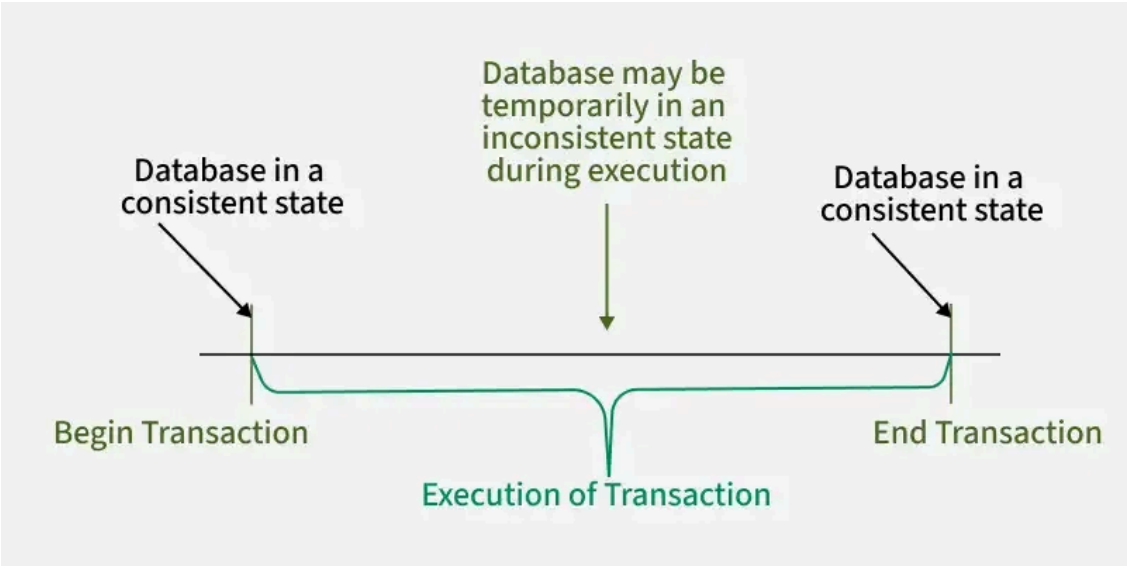
Transaction in DBMS

Last Updated : 12 Mar, 2025

In a Database Management System (DBMS), a transaction is a sequence of operations performed as a single logical unit of work. These operations may involve reading, writing, updating, or deleting data in the database. A transaction is considered complete only if all its operations are successfully executed, Otherwise the transaction must be **rolled back**, ensuring the database remains in a consistent state.

Databases SQL MySQL PostgreSQL PL/SQL MongoDB SQL Cheat Sheet SQL Interview Questions My

A transaction refers to a sequence of one or more operations (such as read, write, update, or delete) performed on the database as a single logical unit of work. A transaction ensures that either all the operations are successfully executed (committed) or none of them take effect (rolled back). Transactions are designed to maintain the integrity, consistency and reliability of the database, even in the case of system failures or concurrent access.



Transaction

All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction.

During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.

Example:

Let's consider an **online banking application**:

- **Transaction**: When a user performs a **money transfer**, several operations occur, such as:
    - **Reading** the account balance of the sender.
    - **Writing** the deducted amount from the sender's account.
    - **Writing** the added amount to the recipient's account.

In a **transaction**, all these steps should either complete successfully or, if any error occurs, the database should **rollback** to its previous state, ensuring no partial data is written to the system.

Facts about Database Transactions

- A transaction is a program unit whose execution may or may not change the contents of a database.
- The transaction 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.

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:

1) Read(X)

A read operation is used to read the value of a particular database element **X** and stores it in a temporary buffer in the main memory for further actions such as displaying that value.

Example:
For a **banking system**, when a user checks their balance, a **Read** operation is performed on their **account balance**:

```
SELECT balance FROM accounts WHERE account_id = 'A123';
```

This updates the balance of the user's account after withdrawal

2) 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.

Example:
For the **banking system**, if a user withdraws money, a **Write** operation is performed after the balance is updated:

```
UPDATE accounts SET balance = balance - 100 WHERE account_id = 'A123';
```

This updates the balance of the user’s account after withdrawal.

3) 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.

Example:
After a successful money transfer in a banking system, a **Commit** operation finalizes the transaction:

```
COMMIT;
```

Once the transaction is committed, the changes to the database are permanent, and the transaction is considered **successful**.

4) Rollback

If an error occurs, the **Rollback** operation undoes all the changes made by the transaction, reverting the database to its last consistent state. 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.

Example:
Suppose during the money transfer process, the system encounters an issue, like insufficient funds in the sender's account. In that case, the transaction is rolled back:

```
ROLLBACK;
```

This will undo all the operations performed so far and ensure that the database remains consistent.

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.

Example:
In a serial schedule, the first transaction completes before the second transaction starts:

1. Transaction 1: Read balance → Update balance → Commit
2. Transaction 2: Read balance → Update balance → Commit

ii) Non-Serial Schedule

To reduce the waiting time of transactions in the waiting queue and improve the system efficiency, we use non-serial 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.

Example:
Transaction 1 and Transaction 2 can be executed in parallel:

- Transaction 1: Read balance → Update balance
- Transaction 2: Read balance → Update balance

Without proper isolation mechanisms, this may cause inconsistencies.

Serializable

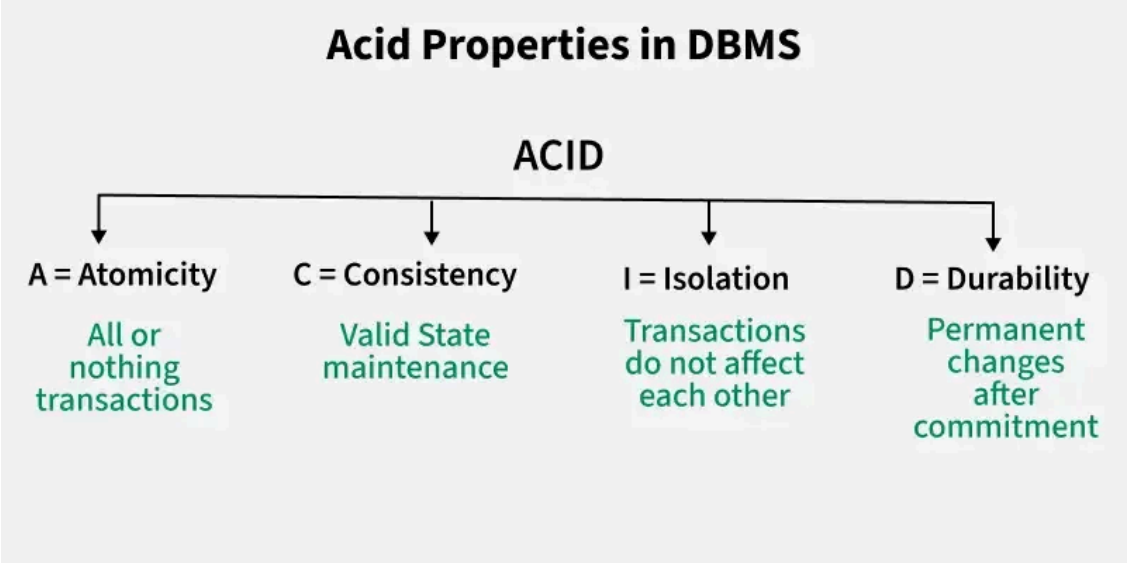
Serializability in DBMS is the property of a non-serial schedule that determines whether it would maintain the database consistency or not. The non-serial schedule which ensures that the database would be consistent after the

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.

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.



ACID Properties

1) 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.

Example:
If a user is transferring money from one account to another, both the **debit** and **credit** operations must be successful, or none should happen. If any step fails, the transaction will be rolled back entirely.

2) 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.

Example:
If an account balance before the transaction is 1000, after a successful transaction of 200, the new balance should be 800, ensuring the database stays in a consistent state.

3) 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.

Example:

- **Transaction 1**: Withdraw $100 from account A.
- **Transaction 2**: Withdraw $200 from account A.

Both transactions should execute as if they are isolated from each other, and their changes should not conflict.

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.

Example:
After transferring money between two accounts, if the system crashes, the changes made by the transaction should still be saved when the system restarts.

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.

FAQs

What is meant by a transaction in DBMS?

> *In DBMS, a transaction is a set of logical operations performed to access and modify the contents of the database as per the user's request.*

What is meant by ACID properties in transactions?

*ACID is an acronym used for the properties of transaction in DBMS. It is used to denote the properties of transactions, i.e. **A**tomicity, **C**onsistency, **I**solation and **D**urability.*

Which operation of transactions ensures the durability property?

In DBMS, the durability of a transaction, i.e. the changes made by it are saved to the database permanently, is ensured by the 'Commit' operation. A transaction is completed only if data is saved using 'Commit' operation. And then, the changes remain durable, i.e. in case of any system failures, the last saved state of the database can be recovered through database recovery subsystem in DBMS.

What is meant by schedules of transactions in DBMS?

When multiple transaction requests are made at the same time, we need to decide the order of execution of these transactions. This chronological order of execution of transactions is called as a schedule of transactions in DBMS. It is mainly of two types, i.e. Serial Schedules and Non Serial Schedules.

What do you mean by serializability in DBMS?

Serializability is the property of a schedule of transactions in DBMS which determines whether the database would be in consistent state or not if the transactions are executed following the given schedule.

Next Article

Transaction States in DBMS

Similar Reads

Transaction in DBMS