UNIT –IV

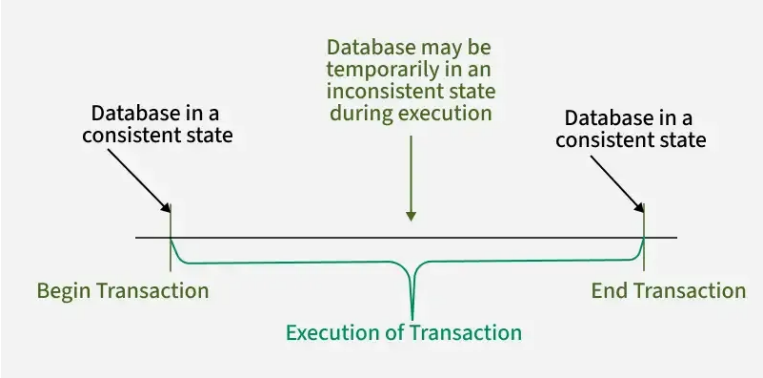
TRANSACTION

Transaction in DBMS

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 .

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 stores updated data from main memory back to the database. It usually follows a read, where data is fetched, modified (e.g., arithmetic changes), and then written back to save the updated value.

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

A rollback undoes all changes made by a transaction if an error occurs, restoring the database to its last consistent state. It helps prevent data inconsistency and ensures safety.

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.

ACID Properties of Transaction

Transactions in DBMS must ensure data is accurate and reliable. They follow four key ACID properties:

Atomicity: A transaction is all or nothing. If any part fails, the entire transaction is rolled back. Example: While transferring money, both debit and credit must succeed. If one fails, nothing should change.

Consistency: A transaction must keep the database in a valid state, moving it from one consistent state to another. Example: If balance is ₹1000 and ₹200 is withdrawn, the new balance should be ₹800.

Isolation: Transactions run independently. One transaction's operations should not affect another's intermediate steps. Example: Two users withdrawing from the same account must not interfere with each other's balance updates.

Durability: Once a transaction is committed, its changes stay even if the system crashes. Example: After a successful transfer, the updated balance remains safe despite a power failure.

Read more about

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. Example: After a successful transfer, the updated balance remains safe despite a power failure.

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

i) Serial Schedule

In a serial schedule, transactions execute one at a time, ensuring database consistency but increasing waiting time and reducing system throughput. To improve throughput while maintaining consistency, concurrent schedules with strict rules are used, allowing safe simultaneous execution of transactions.

ii) Non-Serial Schedule

Non-serial schedule is a type of transaction schedule where multiple transactions are executed concurrently, interleaving their operations, instead of running one after another. It improves system efficiency but requires concurrency control to maintain database consistency.

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Lets take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

As Account

Open_Account(A)

Old_Balance = A.balance

New_Balance = Old_Balance - 500

A.balance = New_Balance

Close_Account(A)

Bs Account

Open_Account(B)

Old_Balance = B.balance

New_Balance = Old_Balance + 500

B.balance = New_Balance

Close_Account(B)

ACID Properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

Atomicity − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

Consistency − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

Durability − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

Isolation − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

Schedule − A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.

Serial Schedule − It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

Equivalence Schedules

An equivalence schedule can be of the following types −

Result Equivalence

If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

View Equivalence

Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.

For example −

If T reads the initial data in S1, then it also reads the initial data in S2.

If T reads the value written by J in S1, then it also reads the value written by J in S2.

If T performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

Conflict Equivalence

Two schedules would be conflicting if they have the following properties −

Both belong to separate transactions.

Both accesses the same data item.

At least one of them is "write" operation.

Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if –

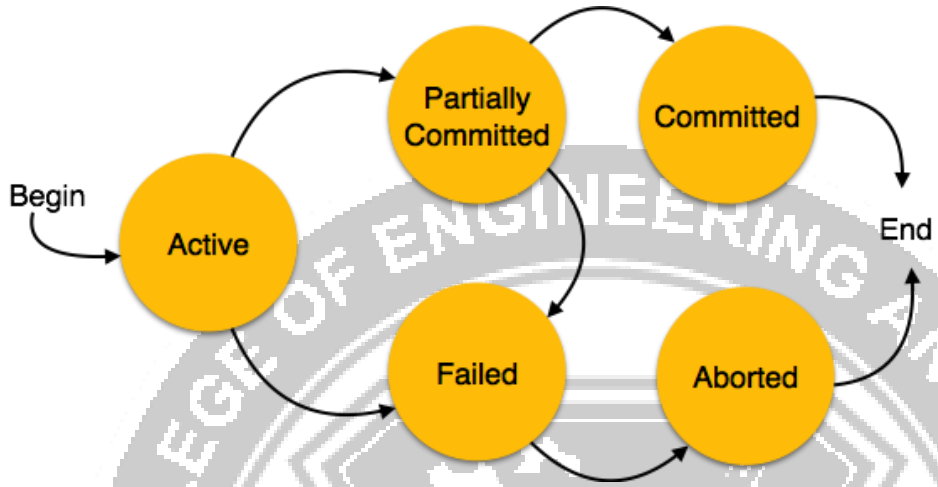
Both the schedules contain the same set of Transactions.

The order of conflicting pairs of operation is maintained in both the schedules.

Note − View equivalent schedules are view serializable and conflict equivalent schedules are conflict serializable. All conflict serializable schedules are view serializable too.

States of Transactions

A transaction in a database can be in one of the following states –



Active − In this state, the transaction is being executed. This is the initial state of every transaction.

Partially Committed − When a transaction executes its final operation, it is said to be in a partially committed state.

Failed − A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

Aborted − If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts –

Re-start the transaction

Kill the transaction

Committed − If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.